Objectworlds : a class of computer-based discovery learning environments

Thesis

Objectworlds: A Class of Computer-Based Discovery Learning Environments

Royston Sellman

Thesis submitted for the degree of Doctor of Philosophy in Educational Technology

Institute of Educational Technology

The Open University

UK

December 1994

Abstract

It is possible to discern a class of Computer-Based Discovery Learning Environments which centre on novel, concept rich, simulated objects and which include simple but general functions with which the objects may be manipulated. This thesis provides a history of this class of environments, which we call *objectworlds*, and we also give them a strict definition. We describe Gravitas, a new objectworld we have built, which allows learners to work with objects that behave like gravitating masses moving in a two dimensional space.

Gravitas contains a powerful *programmable* interface to the objects, in the form of a set of Logo commands, and a functionally equivalent but easier to use *graphical* interface which is controlled by the mouse. We show that the combination of interfaces helps learners to explore the world of these objects more effectively.

We contrast the educational experiences learners are afforded by objectworlds with those offered by two closely related kinds of Discovery Learning Environment: Simulations and Modelling Systems. We also describe a psychological framework which provides a useful way of thinking about the construction of computer simulated objects for discovery learning applications.

Acknowledgements

Mark Elsom-Cook and Tim O'Shea supervised this research. At an early stage Mark corrected my naive view that the progress targets set in our meetings were simply private jokes between us. As is traditional for Mark's students, my thesis plan was worked into a feasible structure during an evening "tutorial" and recorded on a damp de-laminated beermat. Tim offered invaluable criticism and advice at many points. He also read every chapter at least once and acted as an incisive reality check.

My interest in the field was originally stimulated by David Squires, a colleague at King's College London. In particular, he encouraged my conversion from Logo sceptic to zealot, ensuring that I learned to see beyond C, so to speak.

Crucial planning sessions of my work occurred at another traditional Open University venue: the M1 motorway. Diana Laurillard's Excellent Taxi Tutorials take place on the Northbound carriageway most weekday mornings, and Progress Reports are discussed, Southbound, between 5 and 6pm. They are to be recommended for all London-based postgraduates, especially (in view of the very reasonable fare) the impecunious. I am grateful to Diana for her positive reaction to some of my better ideas, and her gentle de-construction of the others.

Many other people have helped my research in one way or another. My friend and fellow student Magnus Moar provided key references and generous hospitality in equal measure. Josie Taylor helped me to obtain the hardware which made Gravitas feasible. Peter Whalley made valuable recommendations about the early Gravitas interface which was, to say the least, user hostile. Ben du Boulay asked difficult questions at the right time and suggested the name. Maria Yannissi and Simeon Yates made comments on sections of the thesis. Ann Blandford did a superb job on my first draft, checking it rapidly, thoroughly and at very short notice. She found some bad mistakes.

Finally, of course, I would have been completely unable to finish this work without the dedicated support of my partner. Thanks Angie.

Table of Figures

Preface

The Aims of this Thesis

The focus of this thesis is a description of a novel computational environment, called Gravitas, which allows learners to explore the behaviour of objects that obey Newtonian laws of motion and gravitation. These gravitating objects, which we call *Massobs*, are a unique feature of the system and they have been designed so that users can manipulate them in very natural ways. In particular, it is easy to create new Massobs, position them on the screen, set them in motion, and then watch their trajectories evolve.

In the thesis we describe the various functions of Gravitas, and the underlying mechanisms which give Massobs their special behaviour. We also show what is special about the learning activities the system can support.

However, the thesis has a broader aim. We wish to make some *general* comments, not just about Gravitas but about systems like it, in order to help others to build similar systems. These remarks though, need a context, and to provide this we have to do two things. First of all, we must show that a distinct *class* of such environments actually exists, and that it is possible to decide whether a particular system belongs to the class or not. Secondly, we must provide a description of their educational character, to convince other developers that building these kinds of system will be worthwhile.

We use the name *objectworlds* for Gravitas and the other systems in the class. They have two main characteristics: a simulated object which is visible on the screen, and a programming language containing commands with which the object may be manipulated or inspected in very general ways. A well known example of an objectworld is Turtle Geometry (Papert, 1972; Papert, Watt, diSessa and Weir, 1979; Papert 1980) - the combination of a small stylised 'Turtle' which has a clearly visible position and heading, and a programming language, usually Logo. Turtle Geometry has been used extensively in mathematics education and it still has a wide following among practising teachers and academic researchers.

We should explain the name *objectworld*, which we have chosen for this class of environments. Another name we considered was *microworld*, as this is the term that Papert uses on many occasions to describe Turtle Geometry

and, by extension, other systems based on a programming language and "manipulable computational objects" (Papert, 1987a). However, over the last twenty years or so, the name microworld has been applied to many other systems, some of which contain neither objects nor programming language, and which are therefore fundamentally different from Turtle Geometry or Gravitas. The name objectworlds emphasises that the systems we are considering are based on objects and also hints at a link to Object-oriented Programming (see for instance Cox, 1986), which is appropriate as we have found the Object-oriented metaphor to be very useful in their implementation. Furthermore, 'world' implies a place for exploration and action, which is exactly what these environments are: "a simplified piece of reality, which you can explore, and [in which] there's no right or wrong" (Papert, 1987a).

With regard to their educational character, objectworlds offer users a variety of discovery learning together with an opportunity to build things. For example, a child using Turtle Geometry may explore the behaviour of the Turtle and discover how to build a program which draws a triangle. Then, with guidance, the child might generalise this discovery into a program which draws polygons. In a similar progression, a user of Gravitas could discover how to make one Massob orbit another and then be led to construct a program which automatically creates planetary systems. From these two examples it should be clear that the character of the central objects defines the particular knowledge domain to which an objectworld applies, while the language, at the cost of learning how to program, offers users flexibility and the possibility of generalising from their discoveries.

We said at the beginning that our main aim is to make some general observations about objectworlds and to provide guidelines for those who wish to build their own. One source of motivation for this wider task is the work of Seymour Papert. Papert has frequently emphasised (see (Papert, 1987a) or the preface to (Papert, 1980)) that he believes the construction of objects like the Turtle, which he calls a *transitional* object because it connects both to a child's sensorimotor knowledge and to deep mathematical ideas, is the most profound possibility that computers can offer education:

> "...an entirely new kind of object - a transitional object between the ones that you can touch and push (like tables and wooden blocks) and the kind of objects that

> you know in science, in philosophy, and in mathematics... This ability to create transitional objects gives us a way of closing the gap between intuitive and formal learning." (Papert, 1987 p88)

We will discuss Papert's notion of transitional objects in more depth in chapters 1 and 6. Briefly, they may be thought of as objects which help to bridge the gap between a child's personal, intuitive knowledge and the abstract concepts of science and mathematics taught at school. Thus a lever or a set of gear wheels, which can be held and manipulated and which display the idea of ratio in a physical form, might be viewed as transitional objects for multiplication and division. Papert believes, furthermore, that a computer - "the Proteus of machines" can be used to engineer them. Recognising that the Turtle itself is not a universal answer, he urges people to build other transitional objects:

> "My concept of how to create a curriculum (and by this word I mean a coherent set of materials to aid learning through the whole school period - and before and after, as well) is to create a network of microworlds, each focussing on different areas of knowledge." (Papert 1987a)

For one reason or another this has not happened. Perhaps a cause is that few educators share Papert's confidence in transitional objects. However, we believe a more powerful deterrent has been a shortage of serious attempts to show how it may be done. This thesis is not just a description of a new objectworld, Gravitas, and its related transitional objects, but is intended to be a source of practical advice and theoretical justification for the approach. According to Papert, (1987b) the educational benefits of objectworlds will not be made apparent purely through the scrupulous examination of one or two examples. We need to see many more of them in use, forming Papert's curriculum network, to properly judge their worth. Nevertheless, if this thesis stimulates an interest in objectworlds on their own merits, without necessarily convincing the reader of the need for Papert's curriculum network, then we will judge it a success. The important thing is that more objectworlds are built.

Gravitas is joined to Logo so that users may write programs which control Massobs. So the question arises as to whether Gravitas is intended for use in the teaching of programming. The short reply is no, but a proper answer requires a digression. We are aware that many studies have been made of children learning to program in Logo, and that some of these studies have

used programming the Turtle as a central activity. All that has changed in Gravitas is the central object. The language is still there and so clearly Gravitas *could* be used to teach programming. However, it was not built specifically for that purpose, but was designed to give children the opportunity to play and work with objects which are easy to position, move and accelerate and yet which connect to profound concepts from physics.

For the most part this thesis avoids the issues which attach to learning to program. In fact, in our studies of Gravitas in use, none of our subjects had any prior experience of Logo, and only two had done any programming at all. We found that the programming component of the tasks the subjects were set was simple enough for them learn "on the fly". For more advanced work with Gravitas though, knowledge of programming becomes an issue, but one which takes us beyond the scope of this thesis. The examples we give in chapter 3 show that even simple Gravitas programs can be educationally rich.

We will finish this preface with a short description of each chapter in the thesis. Overviews will also be found at the beginning of each chapter, and the main conclusions of the research will be summarised at the end.

Chapter 1

The purpose of this chapter is to describe the history of objectworlds, that is, educational computing systems which combine simulated objects of some sort with a programming language equipped with commands to control them. We begin with Logo and the Turtle in the late 1960s and move on to highlight the essential developments which have taken place, such as the introduction of *dynamic* objects like Dynaturtles, which contrast with the ordinary Turtle that lies static between commands. Particular emphasis is put on the difference between two kinds of *implementation* of dynamic objects. Those which, like diSessa's original implementation of Dynaturtles, remove concurrent access to the programming language, and others, such as the various sprite Logos, which do not. At the end of the chapter we present a strict definition of objectworlds.

Chapter 2

In this chapter we describe a new objectworld, called Gravitas, which extends the concept by building on the examples of the past. In particular, Gravitas supports a new kind of dynamic object, called a Massob. Like sprites,

these move across the screen continuously, and may be controlled by Logo commands which the user types in. Massobs however, go beyond sprites by adding a second layer of behaviour: they obey the laws of gravity so that as they move, their trajectories curve in response to the presence of others.

Gravitas also extends the objectworld concept by adding to the programming commands a second method of controlling and inspecting the objects of interest. This is called the *graphical* interface and it combines screen-based buttons and meters with elements of direct manipulation, making it possible to work with Massobs using only the mouse.

Chapter 3

Here we describe Gravitas in use with real subjects, aged between 13 and 18. The purpose is not to show how good Gravitas is for teaching the concepts of physics compared, say, to conventional methods. We believe this would be premature. Our contention is that Gravitas, the combination of Logo and Massobs, opens up a new space of possibilities for discovery learning in physics. Accordingly, the chapter is a report on studies which illustrate the character of this educational space.

We stress two main points. First of all, we have found that users of Gravitas are often surprised by the behaviour of even quite simple systems of Massobs, *which they have themselves constructed.* We show that learners are strongly motivated to resolve these surprises, a process which requires them to think hard about the physical concepts involved. Secondly, we show that the combination of the programmable and graphical interfaces allow students to take on more complex tasks than would otherwise be the case.

Chapter 4

In this chapter we survey the Science National Curriculum to identify areas for which Gravitas based activities seem appropriate and easy to construct. In particular we examine the Statements of Attainment dealing with concepts such as Force, Momentum, Energy and Gravity, and with broader, descriptive knowledge of astronomical bodies and satellites.

We also examine the extent to which the programmable nature of Gravitas opens up wider educational possibilities, such as longer term investigations of a particular concept.

Finally, we describe the way in which Gravitas may have its basic functionality augmented through the addition of procedures written in standard Logo.

Chapter 5

We devote this chapter to setting objectworlds in context. There are many kinds of educational computing systems and over the years researchers have produced several schemes of categorisation. After a survey of some of these schemes we compare objectworlds with the programs we consider to be their closest relatives: Simulations and Modelling Systems. We show what is different about the kinds of learning they support and, in a synthesis at the end of the chapter we suggest that each kind of system is suited to different stages of concept acquisition.

Chapter 6

Seymour Papert has described the Logo Turtle, and objects like it, as *transitional* because they lie conceptually between the things of everyday experience, like tables, chairs, stones and bicycles, and the formal objects, like differential equations and point masses, which science is built upon. He goes on to describe the means by which we manipulate them (forward, back, right and left in the case of the turtle) as *syntonic commands*. However, his comments on both of these topics are spread across several books and papers. In this chapter we bring together in one place Papert's comments about what transitional objects and syntonic commands can offer education. We also look at what two other researchers - Donald Winnicott and Robin Hodgkin, have had to say about transitional objects.

Chapter 7

The seventh chapter of this thesis is where we summarise the main contributions of our research. Principally these are:

(i) The definition of a particular class of educational computing systems, called objectworlds. (Chapter 1)

(ii) The construction of a new member of the class, Gravitas, which instantiates a new kind of transitional object, the Massob. (Chapter 2)

(iii) A study of the learning activities Gravitas can typically support, with an emphasis on the surprising behaviour of Massobs and the scope of simple programs which control them. (Chapter 3)

(iv) A group of example systems and programs which illustrate Gravitas' applicability within the framework of the National Curriculum for Science and which also indicate its wider scope. (Chapter 4)

(v) A comparison of objectworlds with their near neighbours in the spectrum of educational computing: Simulations and Modelling Systems. (Chapter 5)

(vi) A discussion of the psychological foundations of transitional objects and syntonic commands, upon which objectworlds are, in part, based. (Chapter 6)

(vii) A survey of the mathematical techniques which have been used to generate the gravitational behaviour of Massobs. (Appendix A)

In chapter 7 we also outline several directions which future research could take, in terms both of empirical studies to be carried out with Gravitas as it stands, and improvements which could be made to the system.

1 The Origins of Objectworlds

1.1 Overview

This chapter is concerned with describing a class of educational computing systems whose members share two essential characteristics. First, a simulated object, and second, a programming language with commands which allow the object to be manipulated or inspected in interesting and general ways. The *attributes* of the simulated object are of key importance as they determine the educational nature of the system. For instance, in one example of the class, Turtle Geometry, the central object has the attributes of position and heading, and it supports investigations into many of the mathematical concepts we might wish children to learn.

At the same time, the object must have a behaviour with which the learner can readily identify, like the turtle's ability to turn and move. This requirement comes from the Constructivist view of learning (Forman and Pufall, 1988), which considers it important to present new knowledge in a way that learners may easily integrate with what they already know. One might say that these simulated objects are 'engineered' in such a way that they connect both to intuitive knowledge, which the learner already has, and to formal concepts we consider to be important parts of a curriculum.

The importance of the programming language will be highlighted in chapters two, three and four of this thesis but we should make some explanatory comments here. Fundamentally, the issue is about how learners can manipulate the objects of interest in a system. Obviously, computers are not the only places where we can engineer an object to aid the learning of concepts. As another author has written about systems (which he calls microworlds) analogous to those with which we are concerned:

> "Microworlds need not be on a computer... Cuisenaire rods, multibase arithmetic blocks, fraction bars, and Miras and tracing paper, all with their respective rules of manipulation, can also be thought of as examples of microworlds" (Thompson, 1985a)

The learner can configure the objects of these learning tools into arbitrary states, each of which can be thought of as representing a mathematical expression, such as $3x^2=12$. Furthermore, they can carry out *sequences* of

reconfigurations, obeying the rules, to achieve the solution to a particular arithmetic or algebraic problem, as in x=2. All these operations are carried out *by hand.*

The point is that we can engineer different objects on the computer, perhaps ones which cannot have physical analogues, made from wood or plastic or anything else. But how do we allow the learner to manipulate, inspect and work with them? In chapter two we will see that a so-called Direct Manipulation Interface (Shneiderman, 1982; Shneiderman, 1983; Hutchins et al, 1986) can be useful. And at some point in the future, given advances in the domains of Virtual Reality and Visual Programming (Myers, 1986), we may be able to generate realistic illusions of handling computational objects. But for the time being, a practical medium that offers the learner a comparable facility to set up configurations, and then to build sequences of reconfigurations, is a programming language, such as Logo.

Our task then is to describe the history of those systems which have two essentials: an object and a programming language. Before starting, we should agree on a name for these environments. As we noted above, the name microworld has been used for this class of system and indeed, it is the term we would have preferred to use throughout this thesis. However, we will see that microworld has also been used for many other systems, some of which lack the characteristics mentioned above and which do not, therefore, belong in our discussion. In fact, the term has even been used to describe phenomena unconnected with software, such as cognitive states in a child's mind (Lawler, 1979), so there is clearly the need for a new name, to cover our specific area of concern.

The name we will use for environments that do have the essential characteristics is *Objectworld.* Towards the end of this section we will present a definition for the term, a definition that will make sense in the light of the historical notes we give below. Many of the systems we will describe are called microworlds by their designers and we will retain the original terms but, to repeat: not *all* the programs that have been called microworlds qualify as objectworlds under our criteria.

1.2 *Turtle Geometry*

The genesis of Turtle Geometry really starts with JOSS and Lisp, and their associated groups of researchers, in the late 1960s. Researchers at the Rand Corporation in the US were working on the interestingly named Johnniac computer. The letters 'ac', standing for 'automatic calculator' were a popular suffix for machines of that era, and the hardware development team had been led by the famous mathematician, John von Neuman. The software team produced JOSS, the Johnniac Open Shop System (Baker, 1981). JOSS was a simple, general purpose programming language, which was *interpreted*. This meant that users could type programs into the computer in human readable form and run them immediately, while the interpreter concealed the mechanisms by which the human code was converted into machine instructions. Like its contemporary, Basic, JOSS was able to support computations on both numbers and text, and the Johnniac could handle several JOSS users simultaneously by a process known as *time-sharing*. JOSS differed from Basic by offering what were, for the time, friendly responses to the user's errors.

Wallace Feurzeig was shown JOSS by one of the original Rand researchers in 1965, and within a short time his own company, Bolt, Beranek and Newman, had their own version running. BBN were unsuccessful in their efforts to sell time on the JOSS system to engineers but Feurzeig had another plan (his early work is described in Feurzeig, 1969 and 1984). Inspired by the ease of use which interpreters brought to computing, he installed a number of terminals in a school and set out to see if, through programming, children could learn some powerful mathematical ideas. Encouraged by the preliminary results Feurzeig obtained funding to continue his research. He wondered if the children, who had made good progress with a programming language designed for professional scientists and engineers, could do much better with a language designed specifically for education. At this point he was joined by Seymour Papert, from the Artificial Intelligence laboratory at the Massachusetts Institute of Technology (MIT), and the two of them collaborated on developing the new language, which they called Logo.

Naturally, Feurzeig wanted Logo to be conversational (that is, interactive and interpreted), just like JOSS. What Papert brought to the design was the influence of the MIT laboratory, where a group of computer scientists,

mathematicians and programmers had developed a powerful language called Lisp, which was equally suitable for computations involving either symbolic or numeric data. From Lisp, Logo inherited *lists*, a particularly powerful method of ordering data, and *recursive functions*, that is, functions that can call themselves. These two features set Logo (and Lisp) clearly apart from the popular languages of the time like Fortran, Basic and COBOL, and although there are excellent reasons for including lists and recursion in a language (see Abelson and Sussman, 1985, Chapter 1), a large part of the computer science community considered itself to be doing quite well without them. Consequently, although Logo quickly attracted devotees, both in the United States and abroad, it also came in for a measure of criticism.

Feurzeig and Papert continued their research into children learning mathematics through programming. Around 1969, a graduate student at MIT called Mike Paterson suggested that the addition of a small robot, driven by extensions to the Logo language, might make Logo more attractive to children. The robots were quickly built at both MIT and BBN and incorporated into the studies. Small electric stepper motors allowed them to move forwards and backwards, and turn right or left, and a pen was attached so that the device could leave a trace on the surface over which it moved. The robots were given the name 'turtle', according to Papert "in honor of a famous species of cybernetic animal made by Grey Walter, an English neurophysiologist" (Papert and Solomon, 1971).

These floor turtles were very popular with the children, and the researchers began to think about meaningful mathematical activities to use them for. At the same time, advances in computer display technology allowed the Turtle to be simulated on a graphics screen where it was less likely to be trodden on or get its wires tangled. This screen Turtle inherited the same *coordinate independent* relative move commands from its floor-based ancestor (Papert and Solomon, 1971), but it also gained the attribute of *absolute position*, based on the coordinate system of the display. In the context of our history it was an important event. The first objectworld, Turtle Geometry, had been born.

Turtle Geometry was the name given to the realm of activities opened up by this combination of Logo and a small object that could turn and move. Papert and his co-workers continued to investigate the use of the system with

schoolchildren. Its appearance, and some typical activities are shown in Figure 1.1.

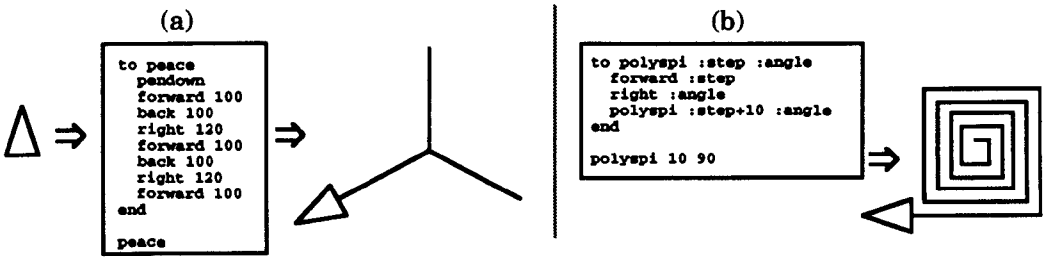


Figure 1.1 Two Logo programs and their effects (After Papert, 1972).

Although the programs shown in figure 1.1 are very simple, they show some of the important features of Turtle Geometry. First of all we can see from figure 1.1a that Logo allows the user to define independent *procedures*, which are then referred to by name. The *definition* of the procedure is the block of text which starts with **to** and ends with **end**. This new procedure is then *invoked* by typing its name, and it causes the Turtle (represented on the screen by a small triangle) to draw the image shown. We can also see that the Turtle commands **forward**, **back** and **right** all take *inputs* which determine how much of each movement the Turtle will make. In figure 1.1b we see that Logo makes it possible for the user to define new procedures which also take inputs. **polyspi** also illustrates the use of recursion (as the term is used in Logo), making a call to itself in the fourth line. What happens when **polyspi 10 90** is typed is that the procedure carries out a step, then calls itself with its first input incremented by 10 and carries out the same step. This process is repeated ad infinitum, producing the growing "squiral" shown, until the user interrupts the procedure.

If Papert had been a conventional computer scientist things might have proceeded in the same way as many other educational computing systems. A typical pattern would have been an initial surge of interest within the development group, followed by experiments using the system within existing school curricula. Upon publication of the results, the original developers move on to more interesting things, and slowly, as advances in the hardware leave the system behind, or the programmers who keep the software running get other jobs, it becomes obsolete and dies. With luck, the expertise gained in building it is passed on to the next generation.

Papert, however, had an unusual background. He had originally trained as a mathematician, but in 1959 he had gone to Switzerland to work with the

famous developmental psychologist, Jean Piaget. Here, for five years, he researched into the nature of children's thought processes. Then, in 1964, he took up a post at MIT. As he puts it:

> "In 1964 I moved from one world to another. For the previous five years I had lived in Alpine villages near Geneva, where I worked with Jean Piaget. The focus of my attention was on children, on the nature of thinking, and on how children become thinkers. I moved to MIT into an urban world of cybernetics and computers. My attention was still focussed on the nature of thinking, but now my concerns were with the problem of Artificial Intelligence: How to make machines that think?" (Papert, 1980 p208)

Obviously, the Turtle is not a thinking machine, but in Papert's view it could have a great deal to do with the way children learn to think about concepts. He began to construct a psychological framework to support his contention that Turtle Geometry, and environments like it, represented an unprecedented opportunity for education:

> "I believe with Dewey, Montessori and Piaget that children learn by doing and by thinking about what they do. And so the fundamental ingredients of educational innovation must be better things to do and better ways to think about oneself doing these things.
>
> I claim that computation is by far the richest known source of these ingredients. We can give children unprecedented power to invent and carry out exciting projects by providing them with access to computers, with a suitably clear and intelligible programming language and with peripheral devices capable of producing on-line real-time action." (Papert, 1970)

He believed he could construct a new mathematics curriculum around Turtle Geometry that would present the accepted mathematical primitives, which many children find alienating, in a much more appealing way. Furthermore, this new presentation of important *formal* mathematical concepts such as points and lines, vectors and differentials, would also involve the learner in thinking about the neglected *informal* concepts, like the very notion of a mathematical system, or problem solving methods. Referring to this he wrote:

> "[We] will describe a new piece of mathematics with the property that it allows clear discussion and simple models of heuristics that are foggy and confusing for

beginners when presented in the context of more traditional mathematics."
(Papert, 1972 p252)

Papert was putting forward quite a radical thesis - a reconstruction of the way mathematics is taught, which would render it less intimidating to beginners and which would, along the way, be teaching them how to make their own discoveries. His 1972 paper was titled, rather polemically, *Teaching Children to be Mathematicians Versus Teaching About Mathematics*. This attitude attracted other workers in the progressive atmosphere of MIT's Artificial Intelligence laboratory and soon a permanent MIT Logo Group was formed.

The Logo group began to investigate Turtle Geometry in two complementary ways. First of all they explored the range of topics, mathematical and non-mathematical, that Turtle Geometry was capable of supporting (see for instance (Papert and Solomon, 1971), (Abelson, diSessa and Rudolph, 1975)), and secondly they studied children using the system in the laboratory and in computer equipped classrooms (Papert, 1970, Solomon and Papert, 1976). Both strands gave rise to fascinating research but for our present purpose, which is to sketch out the history of objectworlds, we will concentrate on the first line of investigation.

At a very early stage members of the Logo group added to the attributes of the Turtle and effectively made a new objectworld. They did this by attaching touch sensors to the sides of the Turtle and adding extra commands to Logo to deal with them. Specifically, these four extensions to Logo - **`fronttouch`**, **`backtouch`**, **`righttouch`** and **`lefttouch`** return the Boolean value **`true`** when a sensor is in contact with something. They then wrote small recursive procedures which gave the touch Turtle new behaviours. Papert called this new domain *Turtle Biology* (Papert and Solomon, 1971) and showed how procedures could be developed to model animal behaviour. Figure 1.2 shows a very simple example which uses **`touch`**, an amalgam of all the sensor extensions mentioned above, added to a screen turtle.

```
to explore
fd 1
rt 1
if touch [rt 120]
explore
end
```

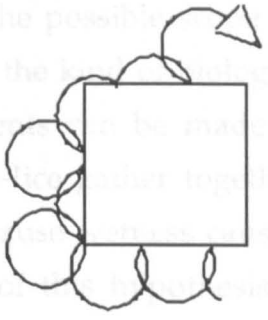


Figure 1.2 A simple Turtle Biology procedure.

The **`explore`** procedure in figure 1.2 behaves something like an animal navigating around an object by feel. The touch Turtle moves forward one step and turns one degree to the right. It will repeat this process forever unless it runs up against an obstacle, in which case it simply turns almost back on itself and then carries on. The effect is that the touch Turtle tends to follow edges. In between encounters the touch Turtle is actually describing a circular path. In fact, without the fourth line (**`if touch [rt 120]`**) explore is simply the classic Turtle Geometry definition for a circle (Abelson and diSessa, 1980). It is unrealistic of course, but the procedure can easily be extended. For instance, if we make the touch Turtle wander randomly, but with a bias to one side, what will happen? Figure 1.3 shows a procedure to do just this:

```
to explore2
fd 1
rt (random 30) - 10
if touch [lt 150]
explore
end
```

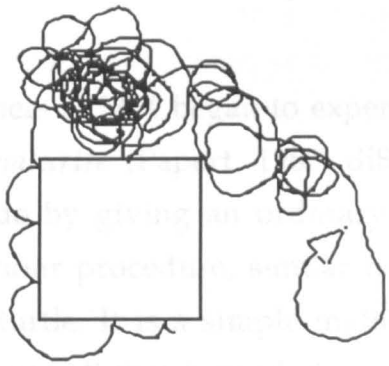


Figure 1.3 A more realistic Turtle Biology procedure.

The path taken by the touch Turtle looks a little more realistic: it crosses and re-crosses its own path and even gets lost. And of course, the procedure may be further modified and experimented with. The second chapter of the book *Turtle Geometry* (Abelson and diSessa, 1980) is devoted to these kinds of activity, and it only begins to show the possible scope of Turtle Biology. Just like Turtle Geometry, it is different to the kind of biology we are used to being taught in schools but strong arguments can be made for its relevance. For instance, biologists believe that wood-lice gather together in moist places not out of an affinity for the wet but because wetness causes them to move more slowly. A conventional examination of this hypothesis requires mathematics of a level too high for many young children, but a few simple Turtle Biology procedures can be the basis of a serious investigation.

1.4 *Dynaturtles*

In the latter half of the 1970s researchers at MIT began to experiment with another object which they called the *Dynaturtle* (Papert, 1980; diSessa, 1982; diSessa and White, 1982). These are made by giving an ordinary Turtle the extra attribute of velocity. A small behaviour procedure, similar to **explore** above, is then used to animate the Dynaturtle. It is a simple matter to add a Dynaturtle to most Turtle Geometry systems. All that is needed is a variable in which to store the velocity (in fact it is better to call it *speed* since the velocity is actually formed from this value and the turtle's heading) and a recursive animation procedure. A typical implementation is shown in figure 1.4. The procedure moves the Turtle forward by a step equal to the value of **speed** and then looks to see if a key is pressed. If the plus or minus keys are pressed then the Dynaturtle is respectively accelerated or decelerated by a small amount. Pressing 'r' or 'l' changes its heading by 10 degrees. The procedure shown assumes **readchar** returns a null value if no key is pressed. In some versions of Logo **readchar** waits until a key is pressed and the procedure would not work.

```
make "speed .1

to dynaturtle
forward :speed
make "key readchar
if equalp :key "+ [make "speed :speed + 0.1]
if equalp :key "- [make "speed :speed - 0.1]
if equalp :key "r [right 10]
if equalp :key "l [left 10]
dynaturtle
end
```

Figure 1.4 A procedure to create a Dynaturtle.

Papert (1980, Chapter 5) describes an evolving sequence of Dynaturtles: The *velocity* Dynaturtle moves smoothly across the screen and continues in a particular direction until a key is pressed (or a function is invoked) to give it a different velocity. An *acceleration* Dynaturtle is similar in that its state is also completely described by its position and velocity. The difference is that this time the operators can only *change* the turtle's velocity by an amount x. The procedure in figure 1.4 effectively creates an acceleration Dynaturtle. The final step in the progression adds the attribute mass and takes us to a *physics* Dynaturtle which behaves "like a Newtonian particle" (Papert, 1980. p.128) and which responds correctly to "kicks" - impulsive forces - that are applied to it.

Andrea diSessa of MIT carried out research using Dynaturtles with children and found that they could be used in an attempt to counter their "Aristotelian" ideas about motion (diSessa, 1982). He observed that most students hold tenaciously to the belief that an object will move in the direction it was last pushed.

> "In view of the striking differences in abilities and style which the students exhibited in their other work, we were greatly surprised to see how uniform their responses to the dynaturtle were. Students seemed to have definite non-Newtonian expectations which were contradicted by the behaviour of the dynaturtle." (diSessa, 1982)

diSessa describes protocols of students using acceleration Dynaturtles which show them confronting their naive ideas and moving towards a "Newtonian" conception, that is, a belief that the velocities of objects are modified by external forces according to Newton's laws of motion.

In another paper (Abelson, diSessa and Rudolph, 1975) some of the scope of the Dynaturtle as an educational tool for physics is indicated. In fact the authors do not mention Dynaturtles at all in this paper, but the theoretical framework used to treat planetary orbits is equivalent. Using "kicks" and velocity vectors (which could be swapped for acceleration Dynaturtles) they prove several of the theorems considered fundamental to orbital mechanics, such as orbit closure and conservation of angular momentum. Admittedly, their treatment is not powerful enough to cover everything important about orbits but, using no more than trigonometry, they are able to derive the standard orbital equation, a result which usually requires some calculus. They contend that achieving such important results without calling up the heavyweight mathematical machinery of the usual analytic approach can give a wider range of students a glimpse of "what doing physics is really like".

1.5 *Objectworlds and Dynamic Objects*

There is a more subtle point that we should try to make clear now. By our convention, when new attributes are added to the geometry Turtle new objectworlds are created. After all, we have said that an objectworld is the combination of a visible object and a programming language which can control it. Therefore a new or augmented object, like a touch Turtle, implies a new objectworld. But there is a problem. The interesting nature of touch and Dynaturtles is generated by the procedures that give the new objects their *dynamic* behaviour. It is this behaviour, rather than the static attributes of the Turtle, that we have shifted our attention to. We are no longer interested in what the touch Turtle is capable of at each step, but in its long term actions. The trouble is, while the procedure which gives the object its new behaviour is running we cannot use the programming language for anything else. Most versions of Logo (in fact, most languages) can only execute one procedure at a time. To run another we would have to interrupt the procedure which is generating the behaviour.

At first sight this might seem unimportant. After all, surely we can arrange it so that any other computations we require, for instance the printing out of the distance travelled by the touch Turtle, are added to the body of the behaviour procedure. But suppose a user wished to add more pieces of program, such as a display of the touch turtle's distance from a fixed point, or a set of controls with which the touch Turtle can be nudged in a particular direction. Soon the behaviour procedure, with its jumble of purposes, will grow large and unclear. To modify the behaviour we would have to search through a procedure cluttered with separate functions. It is better to keep procedures simple:

> "A well written procedure, like a clear, concise paragraph, is devoted to a single topic. You can write procedures that are as small or as large as necessary for the conceptual structure of your program. The important point is that a procedure should perform exactly one identifiable action." (Tatar, 1987 p38)

There are other good reasons for wanting to avoid losing the programming language while a dynamic object is in existence. Imagine, for instance, that we have created an "insect" Turtle whose level of activity depends on the temperature (stored in a variable) of its environment. One could set the temperature variable, as one might set the speed of a velocity

Turtle (see figure 1.4), and observe the results. But it would probably be more interesting to write a second, environmental procedure which simulated the changes in temperature over a typical day. Again, this could be done in the object's behaviour procedure, but at the cost of clarity. Ideally the environment procedure would be separate, and we would ask Logo to run it while the insect turtle's behaviour was maintained independently. Objectworlds are learning environments and clarity is a vital asset. We should avoid forcing learners to understand unnecessary detail.

In this sense, Turtle Biology and the Dynaturtles are restricted objectworlds, because while the dynamic objects are active we lose access to Logo. Later on we will encounter other objectworlds with dynamic objects, some of which continue to allow access to the programming language. In an ideal situation it would be possible to create the dynamic behaviour as a concurrent process, with some means of turning it on and off, and then forget about it. This, as we will see, is just the solution that some systems have adopted.

1.6 *Mindstorms and Microworlds*

As we said above (in section 1.1), objectworlds and microworlds are closely related. Indeed, many of the systems described in this historical survey were called microworlds when they were built. It is only because the term has come to be used for an ever more diverse range of systems, including almost any environment which is not overtly didactic, that it is necessary to introduce the new name. The range of systems we are concerned with in this thesis is quite narrow.

The first people to use the term microworld in an educational context were probably Marvin Minsky and Seymour Papert in 1972 (Minsky and Papert, 1972) although they did not coin the word. At first, their use of the term had little to do with software but instead referred to individual segments of a problem solving domain *and* the cognitive schemata that students interacting with that domain might build. During the 1970s the word continued to be used both for these psychological notions and the pieces of software (turtles, touch turtles, Dynaturtles and so on) that experimenters were using with students. However, with the publication of the book *Mindstorms* (Papert, 1980) the notion of a computer-based microworld was made firmer.

At this time, around 1980, Papert had in mind a "world with its own set of assumptions and constraints" (Papert, 1980 p117). Discussing the concept he writes of "a computer based interactive learning environment where the prerequisites [for learning] are built into the system and where learners can become the active, constructing architects of their own learning" (Papert, 1980. p122). In the context of *Mindstorms* the verb "construct" has two important connotations. First it indicates Papert's belief in the Piagetian idea that learners construct their own understandings by *assimilating* new information and *accommodating* it within their existing knowledge. Second, it emphasises that he sees the construction of procedures and programs as the central activity for children using microworlds.

Papert goes on to stress the importance of progressions of microworlds, such as the sequence mentioned above in section 1.4, which starts with the geometry Turtle and moves via the velocity and acceleration turtles to the Newtonian Turtle. In fact, this progression is central to Papert's concept of microworlds - it accords with two of his "mathetic principles" (Papert, 1980 p.120). First, each new environment is clearly built on something already

learned, and second, each environment is explorable. The idea is to allow children to explore and build in simpler versions of "official" maths, physics, or biology.

Papert's book *Mindstorms* is also where he introduces the term 'transitional object', a concept which we will see later is of great relevance to objectworlds. He explains that an early childhood fascination with gears gave him a hook onto the mathematical ideas he was taught at school. The gears acted as a transitional object which helped him on the way to an understanding of the abstract objects of arithmetic and algebra. We will have more to say on the nature of transitional objects in chapter 6 but some words of explanation are in order here.

In Papert's view a transitional object acts as a stepping stone between a child's personal, intuitive knowledge and the formal concepts we expect her to learn. Papert believes that this gap between a child's sensorimotor schemata and the abstract concepts of science can be bridged by the right transitional objects. Furthermore, he believes that the computer - "the Proteus of machines" can be used to engineer them. As an example of a transitional object Papert cites the Logo Turtle: the commands which make it turn and move are easily apprehended by children, yet they nevertheless connect to powerful mathematical ideas. Similarly, the Newtonian Turtle is manipulated by functions which mirror our everyday experience of pushes and shoves, but it shares important properties with the formal concept from Newtonian physics called a mass particle.

So, it is in fact the *combination* of transitional objects and easily understood commands to control them which is crucial. Papert recognises this. The right combination can foster the kind of learning which he calls *syntonic* - a term "borrowed from clinical psychology" (Papert, 1980 p63). An example he gives is the Turtle circle:

> "Sooner or later the child poses the question: "How can I make the Turtle draw a circle?" In LOGO we do not provide "answers" but encourage learners to use their own bodies to find a solution. The child begins to walk in circles and discovers how to make a circle by going forward a little and turning a little. Now the child knows how to make the Turtle draw a circle: Simply give the Turtle the same commands one would give oneself. Expressing "go forward a little, turn a little" comes out in Turtle language as REPEAT [FORWARD 1 RIGHT TURN 1]. Thus we

see a process of learning that is both ego syntonic and body syntonic." (Papert, 1980 p206)

The relative move commands of the Turtle - `forward`, `back`, `right` and `left` are body syntonic in contrast to say `setxpos 100`, or `setheading 270`, which require an understanding of ideas of coordinate and angle. We will find more syntonic commands in the objectworlds we describe in the rest of this chapter.

The paragraphs above represent a brief and selective summary of some of the ideas in *Mindstorms*, and we will expand on them in chapter 6. However, before we move on, we should make some general comments about the book, as it achieved great popularity and was influential in keeping interest in Turtle Geometry and Logo programming alive. First of all, Papert presented his views in a deliberately iconoclastic style. He did not want to compromise with existing practice for the use of computers in education:

> "Much of the book is devoted to building up images of the computer very different from current stereotypes. All of us, professionals as well as laymen, must consciously break the habits we bring to thinking about the computer... It is not true to say that the image of a child's relationship with a computer I shall develop here goes far beyond what is common in today's schools. My image does not go beyond: It goes in the opposite direction." (Papert, 1980 p.5)

Secondly, anyone who took up Papert's opinions had to do so as something of an act of faith. He did not provide any of the usual statistical evidence for his claims, just examples and a few verbal protocols. This is not to detract from the book, for it was done intentionally. Papert strongly believed that the usual methods were too slow for the introduction of so radical an approach. To sum up his position he once gave an interesting metaphor:

> "This paper is dedicated to the hope that someone with power to act will one day see that contemporary research on education is like the following experiment by a nineteenth century engineer who worked to demonstrate that engines were better than horses. This he did by hitching a 1/8 HP motor in parallel with his team of four strong stallions. After a year of statistical research he announced a significant difference. However, it was generally thought that there was a Hawthorne effect on the horses... the purring of the motor made them pull harder" (Papert, 1970).

1.7 *Abelson and diSessa: Turtle Geometry*

Another important book to be published in 1980 was *Turtle Geometry* (Abelson and diSessa, 1980). Harold Abelson and Andrea diSessa had worked with the MIT Logo group since the early 1970s and their book was both a synthesis and an extension of that work. In effect they present a mathematics course unlike any other. A practical working out of the vision for a new approach to mathematics teaching which Papert first put forward in his paper *Teaching Children to be Mathematicians Versus Teaching About Mathematics* (Papert, 1972). Abelson and diSessa shared Papert's two main beliefs: that the best way to introduce the computer to education is as part of a radical change rather than as a new way of doing the old things, and that computers can give learners the experience of doing science rather than just learning about it:

> "This book is a computer-based introduction to geometry and advanced mathematics at the high school or undergraduate level. Besides altering the form of a student's encounter with mathematics, we wish to demonstrate a curriculum that shows the computational influence in its choice of ideas as well as in its choice of activities...Most important in this endeavor is the expression of mathematical concepts in terms of constructive, process-oriented formulations, which can often be more assimilable and more in tune with intuitive modes of thought than the axiomatic-deductive formalisms in which these concepts are usually couched" (Abelson and diSessa, 1980 pxiv).

In fact the book is pitched at quite a high level, aiming more at the first year undergraduate than the high school student. Also, many of the topics one would expect to see in a traditional mathematics course, such as differential calculus or trigonometry, are missing, and less common areas such as the mathematics of growth and the topology of curved surfaces are included. We have already touched on an example of how different the techniques they employ within these topics are: the touch Turtle and its associated behaviour procedures described in section 1.3. In chapter two of their book Abelson and diSessa expand at some length on the subject of modelling animal behaviour.

Another example of their approach is worth discussing here, as it concerns a topic that *is* a component of most university mathematics or physics curricula: Einstein's Theory of Relativity. A standard course on relativity (for instance Gettys, Keller and Skove, 1989) begins by setting up the mathematical machinery of four-vectors (that is, vectors which have

components in four dimensional *space-time*: the three spatial dimensions of everyday experience, combined with the dimension of time) and then uses it to construct a set of transformation equations which accord with Einstein's fundamental postulates: that the laws of physics are the same, and that the speed of light, c, is a constant, for all observers, regardless of their relative motion. These transformation equations form the basis of *special relativity* and they allow the derivation of a number of quantitative results, such as the equivalence of mass and energy (the famous relation $E=mc^2$), and the slowing down of clocks, which hold in cases where the relative motion of observers is uniform. Next, a new mathematical apparatus, called tensor calculus, is introduced and used to extend the theory to the *general* case, where observers may be accelerating with respect to one another, or may be situated in non-uniform gravitational fields. Again, important quantitative results can be derived, such as the amount by which light rays are bent near massive objects. Finally, to include gravitation between masses we are led to the notion of curved space-time:

> "To give an objective description of the effects which had hitherto been attributed to a force of gravitation, Einstein found it necessary to think of spacetime as curved... There is a special curvature in the presence of matter, although we must not say that the matter is either the cause or the effect of the curvature." (Jeans, 1947)

Under this picture we are led to an enormous conceptual shift. Instead of explaining that the Earth, for instance, is constrained to move in an elliptical orbit around the Sun by the force of gravity, we say that it is simply travelling along the shortest possible path between points in the curved space-time associated with the Sun.

The concepts of relativity, especially curved space-time, are very difficult to understand, and the exposition we have just given is too brief to be much help. However, the important point is that it does show the structure of the story that is usually told about relativity. In contrast, the approach taken in *Turtle Geometry* begins with the construction of a new Turtle which behaves as though it is moving not on a flat plane but on the surface of a sphere. Some trigonometry has to be introduced to do this, but the authors quickly move on to explore the *behaviour* of the new Turtle, and thereby the geometry of curved surfaces. The next step is the introduction of metaphors to give

learners a handle on the concept of curved space, and these metaphors are used in the construction of an object which moves as though its containing space is curved. The student is allowed to get a feel for curved space by manipulating this relativistic Turtle.

Abelson and diSessa do not claim that their approach renders the concepts of General Relativity easier to learn. They are still difficult and somewhat counter-intuitive. Nor is it particularly important that the amount of mathematics used by the *Turtle Geometry* method is smaller. Sooner or later the mathematics would have to be introduced to allow the derivation of important quantitative results, although perhaps many would agree that postponing the heavyweight analysis may mean that less students are deterred. The main innovation of Abelson and diSessa's course is that from the outset it involves the learner in building new computational objects (in the book they are called 'simulators') which behave relativistically, and then uses these objects to explore the major qualitative features of particle motion in curved space.

Toward the end of the 1970s small personal computers began to appear on the market and find their way into the educational system. One of these computers, the Apple II, had the facility for graphics and before long a version of Logo was developed for it. This was quite a breakthrough as for the first time Turtle Geometry could be used by ordinary educators who, unlike the Logo pioneers, had no access to powerful computers.

The Apple II was designed as a machine for computer hobbyists. There were however other affordable computers which owed their existence to the market for computer games and this circumstance led almost by chance to another objectworld.

Two computers in particular, the Texas Instruments TI 99/4A and the Atari XL, contained extra circuitry designed to handle the animation of small graphics on the screen. This circuitry was included to make it easier for programmers to create the fast moving alien invaders that were deemed essential for a popular game. When, to broaden the appeal of their machines, the manufacturers decided to produce versions of Logo for them, it was natural to include commands in the language that could control these graphics. They became known as *sprites*.

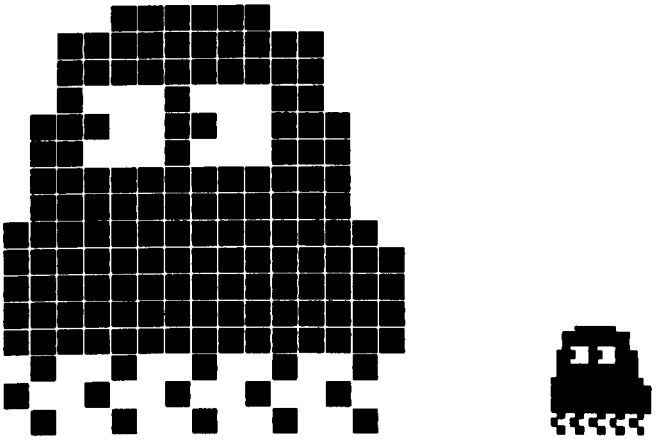


Figure 1.5 A typical sprite, enlarged and approximate actual size.

One set of commands built into TI Logo allows users to define the shape and colour of up to 25 sprites using a grid based editor. Other commands set the heading and the speed of the sprite. The extra electronics take care of the animation, moving the sprites smoothly across the screen, while the computer's main processor is left to run Logo unhindered. Sprites plus Logo, therefore, form a new objectworld with dynamic objects and continuous access

to the language. In fact, a sprite's behaviour is just like that of the velocity Turtle described in section 1.4, but since this behaviour is independently maintained as a concurrent process, the user retains access to Logo. Sprite cards soon began to appear for other machines, notably the Apple II, and extra facilities evolved such as the ability for sprites to detect collisions with one another.

An early member of the Logo community, Robert Lawler, describes an environment built within a sprite objectworld which he calls the *Beach microworld* (Lawler, 1982). This system, using procedures written in Logo, allows the user to construct on the computer beach scenes containing named, dynamic, manipulable objects which resemble things like the sun, clouds, a sailboat and a rider on a horse. They are continuously visible and can be controlled through simple commands like *paint*, *turn* and *zoom*, which set their colour, direction and speed. These objects are implemented as sprites, and Lawler gives some details of the Beach microworld's use as a reading aid for very young children. The main point he makes is that a young child, a pre-reader, can learn to type in the commands that control the sprites without at first understanding them. Then, as action is matched with result, the learner gains a vocabulary. Lawler reports what happened when he allowed one of his own children to use the Beach microworld:

> "For Peggy, the learning of reading and the learning of writing have been synchronized (as speaking and interpreting speech are for the toddler). She learned to read her 30 word vocabulary by learning first to "write", i.e., key the words on the computer terminal. Writing was an essential part of controlling the computer microworld that engaged her." (Lawler, 1982 p146)

Lawler's findings are mainly anecdotal but they do indicate some of the scope for sprite objectworlds. However, as the computer market diverged into games machines and machines for business and education, sprites began to die out.

1.9 *Microworlds that are not Objectworlds*

By the end of 1980 the word 'microworld' was closely linked to the work of Papert's Logo group at MIT, but it was being used in quite a loose way. For instance, Papert himself uses it to mean both the discovery rich environments created by programmers and teachers, such as Turtle Geometry and Dynaturtles, and the smaller constructs built by children playing within those environments:

> "Working with computers can make it more apparent that children construct their own personal microworlds. The story of Deborah at the end of chapter 4 is a good example. LOGO gave her the opportunity to construct a particularly tidy microworld, her 'RIGHT 30 world.'" (Papert, 1980 p162)

Besides the Beach microworld mentioned above, Lawler also discusses two other pieces of software which he calls microworlds (Lawler, 1982 p146). The first, called **polyspi**, deals with the well-known Logo procedure (already shown in figure 1.1) which produces an infinite variety of spiral patterns as it is executed with different inputs (see Figure 1.6). The patterns are drawn by the Turtle (hidden in the diagram) moving in response to the forward and right commands. The recursive call at line 4 causes the procedure to carry on drawing indefinitely or until the user interrupts the process.

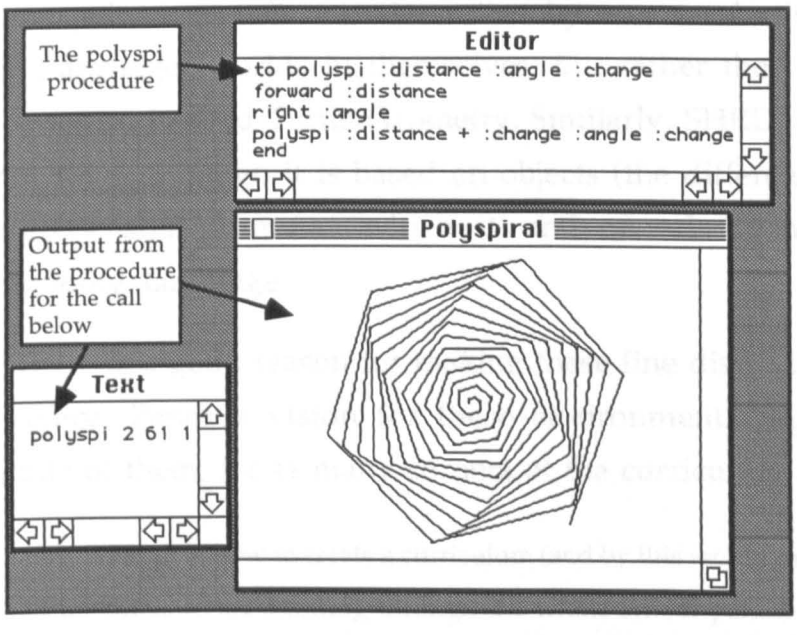


Figure 1.6 The **polyspi** procedure.

The other piece of software described by Lawler is a system for graphing functions and parametric equations. Inputs to the Plotting microworld are

equations which the system then scales over a range of the independent variable(s) and plots on the screen. Arbitrarily complex functions may be used and Plotting demonstrates a neat way to support mathematical modelling activities. Its simplicity illustrates the power of the Logo language.

At the same time as Lawler was writing, Paul Goldenberg (Goldenberg, 1982) gave a rather vague definition of microworlds. So vague, in fact, that it could include almost any piece of interactive software. He writes - "...a microworld is a well defined, but limited, learning environment in which interesting things happen and in which there are important ideas to be learned." Such a definition would not help anyone wishing to build a microworld but the otherwise interesting article does point out some links to the domain of Artificial Intelligence - Goldenberg describes the SHRDLU program (Winograd, 1972) in which a set of different shaped blocks can be arranged in relation to each other through the use of natural language-like commands. SHRDLU's ability to understand commands given in English is impressive and was obtained through a substantial programming effort, but was only rendered possible by inventing an extremely simple domain about which discourse can take place. SHRDLU can accept sentences such as "Place the sphere on the block that is next to the pyramid" but its conversational repertoire is by most standards limited.

The point we wish to make is that by our standards, neither Polyspi or Plotting are objectworlds in themselves, but rather they are activities *within* the objectworld called Turtle Geometry. Similarly, SHRDLU does not meet our criteria, since although it is based on objects (the different shaped blocks) the vocabulary and grammar which it understands is not equivalent to a *programming* language.

There is a good reason for making these fine distinctions. As we noted in the Preface, Papert's vision for these environments requires there to be a multitude of them, for as many corners of the curriculum as possible:

> "My concept of how to create a curriculum (and by this word I mean a coherent set of materials to aid learning through the whole school period - and before and after, as well) is to create a network of microworlds, each focussing on different areas of knowledge." (Papert 1987a)

One of the aims of this thesis is to offer guidance to those educators who will build these systems and this requires a clear notion of what they are. What we are trying to do is show that although the term 'microworld' has been diluted over the years, there does exist a discernible family of environments, containing objects and a language, for which various researchers have made special educational claims. If we get the terms and definitions right, as we intend to by the end of this chapter, our task will be made easier.

1.10 diSessa and White

We briefly mentioned diSessa's experiments with Dynaturtles in section 1.4. In a set of papers published in the early to mid 1980s Andrea diSessa and Barbara White describe various implementations of Dynaturtles and they report their experiences of using them with students. diSessa gives their research a context by describing as "Aristotelian" the students' preconceptions about motion: "One might characterize early stages of students' work as the confrontation of an essentially Aristotelian theory of physics with a Newtonian reality." (diSessa, 1982) He goes on to explain his use of the term Aristotelian:

> "In using the term Aristotelian physics, we generally mean to impute a definite but non-Newtonian stance to our subjects. More specifically, Aristotle's theory of 'violent' (forced) motion is very close to the expectations exhibited by our subjects, specifically with respect to the lack of concern for the effect of previous motion in predicting the results of a force." (diSessa, 1982)

In their empirical work, (diSessa and White, 1982; diSessa, 1982) they use an *acceleration* Dynaturtle controlled by a *kick* command, and the environment they give their students, in the terms of our discussion in section 1.5, is a restricted objectworld. The Dynaturtle is driven by a recursive procedure, analogous to that of figure 1.4, which moves it smoothly across the screen. The user is able to apply kicks, which act in the direction of the turtle's heading, and which modify the turtle's velocity. The tasks put to the subjects take the form of a series of games, each of which involves guiding the Dynaturtle towards a target using turns and kicks. Figure 1.7 shows one of the simplest games and illustrates a conflict between expectation and reality which diSessa and White found typical of most of their students.

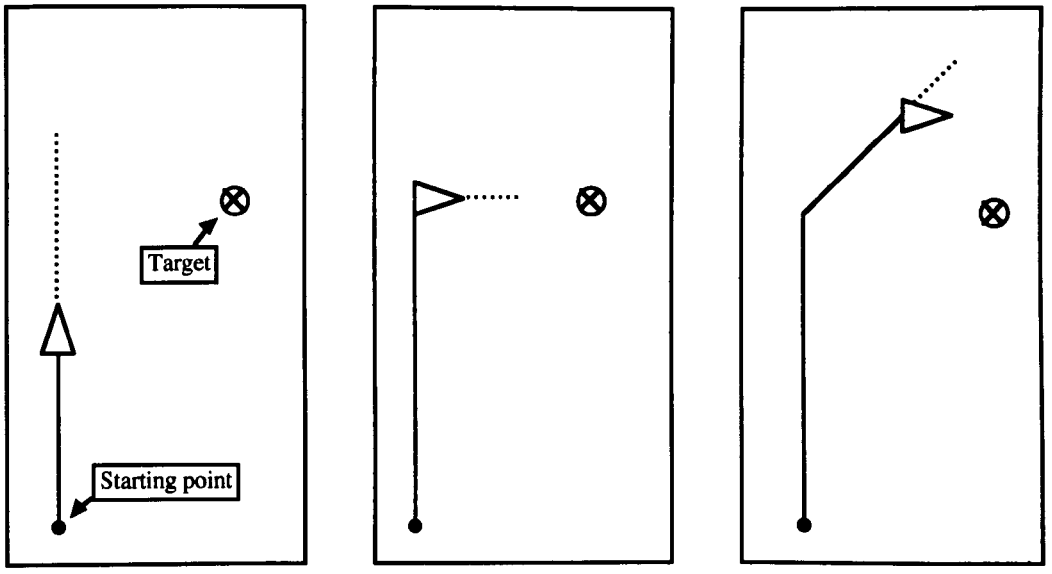


Figure 1.7 (a) after first kick (b) *expectation* after right 90, kick (c) actual motion after right 90 kick

The students tried various strategies to solve this problem. The one shown in figure 1.7, which obviously fails, is called by diSessa the "Aristotelian corner" strategy, since it seems to be based on the assumption that objects travel in the direction they were last pushed. They found that "With time and practice, the feedback from the microworld allowed the students to gain a better understanding of how forces should affect the motion of an object" (diSessa and White, 1982). One successful technique, which was discovered by several students, diSessa calls the "Newtonian corner" strategy (figure 1.8). In this, the Dynaturtle is started with a single kick, then turned through 180 degrees and stopped with another single kick. Finally, the Dynaturtle is turned through 90 degrees to point at the target and then sent to its goal with one more kick.

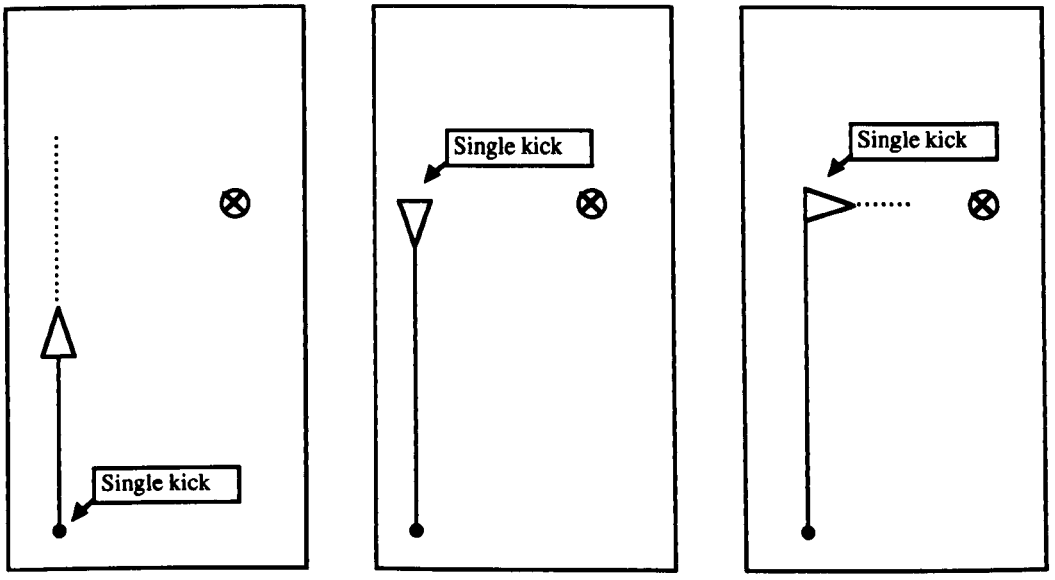


Figure 1.8 The Newtonian corner strategy (diSessa, 1982)

White concludes that games such as these have a significant positive effect on students' ability to solve simple force and motion problems and that this implies an improved intuitive understanding of "both the implications and the applications of Newton's laws of motion." (White, 1984)

1.11 Groen

In 1984, an educational psychologist, Guy Groen addressed what he considered to be "some problems due to the lack of an adequate theory that arise in attempting to evaluate current research on Logo." (Groen, 1984). To appreciate the relevance of Groen's remarks to our history of objectworlds we must review his comments on the Logo based research of that time. His complaint is that although Papert's discussion of microworlds, and what they can do for education, is radical in nature, it is also descriptive and anecdotal. On the other hand, the traditional evaluative techniques applied in some research on Logo lack a theoretical framework through which their results may be interpreted. He puts it quite clearly:

> "Most studies have tended to fall into two extreme categories. The first consists of extensive observations which are then used, in an informal fashion, to provide anecdotes that illustrate some aspect of the Logo approach. The second consists of studies in the tradition of educational evaluation, in which some hypothesis about the outcome of students' interaction with the Logo environment is tested by collecting behavioral measures and subjecting them to appropriate statistical analyses. In both cases, the problem is that we do not know why the observed results occurred. This can only be done in terms of a theoretical framework." (Groen, 1984 p49)

Groen proposes that the rather vague notions in *Mindstorms* should be forged into something precise and testable - a *theory* of microworlds. His programme for achieving this begins by formalising some relevant parts of Piagetian theory (in particular the notion of *structure*, that is, the network of states and transformations that for Piaget characterise a knowledge domain) and combining them with a *frame* notation for modelling the knowledge that students are expected to use. The frame concept is due to Minsky:

> "A frame is a data-structure for representing a stereotyped situation like being in a certain kind of living room or going to a child's birthday party. Attached to each frame are certain kinds of information. Some of this information is about how to use the frame. Some is about what one can expect to happen next. Some is about what to do if these expectations are confirmed." (Minsky, 1975)

Groen goes on to complement the psychological aspects of his design for a theory by prescribing some desirable features of the computational

environments which he, in the same sense intended by Papert, calls microworlds and which we would call objectworlds:

> "A microworld is a structure with certain additional properties. The two most important are:
>
> 1) A transformation can be undone to go back to the previous state.
>
> 2) There should exist mappings (in the precise mathematical sense of the term) to other structures that are representations of concrete actions in the real world." (Groen, 1984)

This is where Groen's work has most relevance to the design of objectworlds. His definition of microworlds is interesting for several reasons (not least of which is his recognition that they actually *require* defining). The first of his "properties" however, seems to be a derived feature of our concept of objectworlds. Because we insist that the programming language contains commands which can both read and alter the state of the central object, it is easy to arrange that the state *before* any operation is preserved in a variable and restored at will. It could be, however, that Groen is thinking of the symmetry of the normal Turtle Geometry operators. Here, **`forward 100`** is negated by **`back 100`**, and **`right 30`** is cancelled out by **`left 30`**. But many useful operators, for instance **`setposition`**, do not have inverses, and our general method of achieving the first property, which Uri Leron calls *conjugacy* (Leron, 1985), is therefore a more powerful notion.

Groen's second property concerns the nature of the central objects. He is saying that the attributes of the microworld object should correspond exactly to the formal theories about the real world that we want children to learn. This is an important point because it implies that we should not present learners with "broken" objects, for instance a mass particle that obeys an inverse *cube* force law. Some designers have done this in other types of environment (Spensley et al, 1990; O'Shea and Smith, 1987), reasoning that by showing users the consequences of incorrect laws or theories, they will discover and appreciate the correct ones. While not ruling out this approach for all kinds of learning environment, Groen is saying that it is inappropriate for microworlds. In fact, common sense leads to a similar conclusion: The central object is engineered so that learners may identify with it easily, and we provide syntonic

commands to assist this process. It would be recondite to go to such lengths for something which is formally wrong.

Groen's work represents a definite step forward towards principled design of the environments we call objectworlds. However, it still leaves the prospective system builder short of guidance on what the software should look like and what functionality it should possess. Furthermore, despite his claim - "A definition along these lines is sufficient to distinguish between microworlds and non-microworlds" (Groen, 1984), the definition is not powerful enough to distinguish between an objectworld and, for instance, a simulation program such as Interactive Physics (Knowledge Revolution, 1989). So, when we come to our own definition we will take Groen's recommendations on board but strengthen them to meet this demand.

1.12 *Squires and McDougall*

Two other researchers of this period, intent on providing useful methods for microworld designers, were David Squires and Anne McDougall. They present what they term an *operational* definition:

> "A computer based microworld is a conjunction of clearly stated primitives enabling transformations of the state of an object (or objects) whose attributes are derived from a fundamental concept, and a set of programming constructs." (Squires and McDougall, 1986)

This formulation has influenced our own definition of objectworlds, as will become apparent in section 1.18. To illustrate their definition Squires and McDougall give two figures:

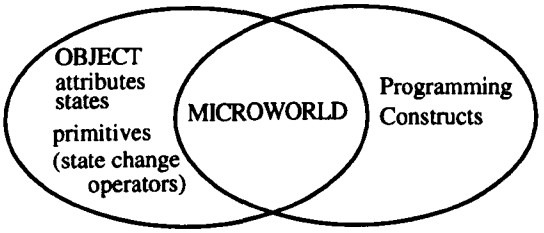


Figure 1.9a The Definition

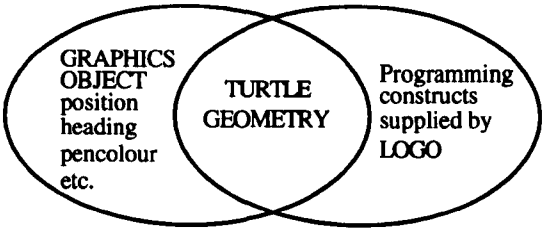


Figure 1.9b An Example - Turtle Geometry

Although they are very positive about the microworld approach in general, Squires and McDougall do urge caution: "Many of the concept areas we have considered, while being rich in interesting ideas, imply microworld objects which are extraordinarily difficult to implement on currently available computing equipment." (Squires and McDougall, 1986) What they have in mind is the fundamental limitation, which we have already mentioned, that to create an object with *dynamic* behaviour usually means giving up continuous access to the programming language.

It is easy to think of examples: for instance, imagine a group of "molecule turtles" which have the same behaviour as the perfectly elastic spheres postulated by the Kinetic Theory of Gases (Jeans, 1967). A molecule Turtle moves with a constant velocity until it collides with another molecule, or the sides of their container. All the collisions are supposed to be perfectly *elastic*, that is, they preserve both momentum and kinetic energy. We can build this behaviour into an ordinary Turtle by the use of a procedure analogous to the Dynaturtle of figure 1.4. If our version of Logo offers multiple turtles (as many versions do) then we can have a group of molecules. But to get them all to

move at the same time we need to arrange for all the behaviour procedures to be run at the same time. In Logo there are 'tricks' we can use to achieve this rudimentary parallelism, but these tricks are far from straightforward and they require that the behaviour procedure can be broken down into simple steps. One such method, called **execute.together**, is described in *Turtle Geometry* (Abelson and diSessa, 1980 p70).

But now imagine that we wish to write a program that takes the molecules through some interesting thermodynamic process, say an isothermal compression followed by an adiabatic expansion. Obviously, it would be possible (though hard) to include the procedure to do this in the **execute.together** process, but once again we have drifted far away from the clarity that must be one of our goals in the design of an educational environment. A better solution to this problem is needed, one which takes us closer to the understandability of objects like sprites. As we shall see below, there are examples of systems which have moved in this direction.

Squires and McDougall go on to describe the application of their operational definition in the building of two exemplar microworlds (also described in Squires and Sellman, 1985). The second of these is of particular note. It is centred around two objects which are continuously represented on the screen: a grid of cells which each contain a numeric value, and an augmented Turtle, called a *Field Turtle*, which moves over the grid and is sensitive to the cell values. The behaviour of the Field Turtle is governed by a user defined rule, which can be set up to mirror a conservation law or a polarity rule, or some other interesting concept from Physics.

The Field Turtle represents a new dynamic object but, as with the Dynaturtle, it gives rise to a restricted objectworld since the behaviour procedure removes access to Logo from the user while it is running.

1.13 Boxer: Dynamic Objects, Definable Behaviour.

In the early 1980s, members of the Logo group at MIT began to develop a new system called Boxer (diSessa, 1986a; diSessa and Abelson, 1986; diSessa, 1986b), which offers the user a computational environment that is radically extended with respect to most other interactive systems. Boxer is rather difficult to describe in a document, but the effort we make here will be worthwhile because the system is such a clear step forward in the evolution of objectworlds

In Boxer the preparation of interactive texts and graphics is seamlessly integrated with the construction and execution of programs. The environment is built upon the organising metaphor of *boxes*, which are rectangular windows on the screen within which the user may read and edit text or programs, or watch computations get carried out. Boxes can be any size and will normally be contained by other boxes. The only uncontained box is called the *world*, and its size equals the whole of the screen. To complement the box metaphor the system holds to another organising principle: that everything that is in the Boxer world is always on the screen. To do this the various boxes are sized, either by the user or by Boxer's intelligent *redisplayer* to reveal the appropriate amount of detail.

diSessa and Abelson call the two principles described above *naive realism* and the *spatial metaphor*:

> "Naive realism is an extension of the 'what you see is what you have' idea that has become commonplace in the design of text editors and spreadsheets, but not for programming languages. The point is that users should be able to pretend that what they see on the screen is their computational world in its entirety." (diSessa and Abelson, 1986 p861)

> "The spatial metaphor encourages people to interpret the organisation of the computational system in terms of spatial relationships. Using a Boxer system is like moving around in a large two-dimensional space. All computational objects are represented in terms of boxes, which are regions on the screen that contain text, graphics, or other boxes." (diSessa and Abelson, 1986 p860)

On one level, Boxer can be viewed simply as a hypertext system (Nelson, 1967), but one in which the task of navigating around the hyperdocument is made

easier by the fact that the whole of the hyperdocument is on the screen all the time.

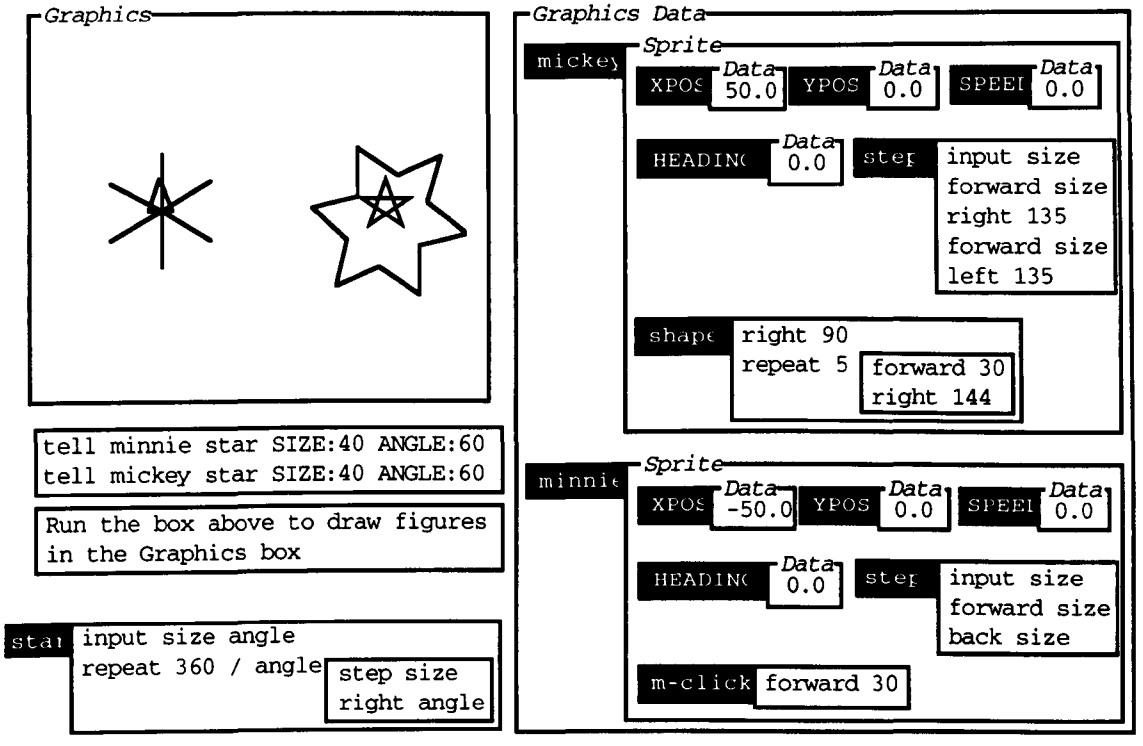


Figure 1.10 A typical Boxer screen (after diSessa and Abelson, 1986)

But to view Boxer only in this way is to miss an entire side of its character. The hierarchy of boxes can also be seen as a visual metaphor for the scoping mechanisms of structured programming languages, and if the text in the boxes conforms to the appropriate syntax rules (Boxer's syntax is very similar to Logo's) then they can be run as a program. In this sense, Boxer is an innovative and powerful new objectworld.

Figure 1.10 shows the description of a Boxer computation and its result. The images shown in the top left box were created by running the commands that appear in the box beneath it. In turn, these commands rely on the processes and parameters contained in the other boxes. It is possible, and perhaps enlightening, to use familiar programming terminology to describe the various items in figure 1.10. For instance some of the boxes have an italicised label interrupting the top left of their frame. These labels mean that the boxes are *typed*, that is they can only contain certain kinds of things. Thus the box labelled `Graphics` can only contain drawings. Some of the boxes, like **`star`**, have names in the black tags at their top left, and these correspond to the named *procedures* (in Boxer named procedures are called *DOIT* (i.e. *do it*) boxes) and *variables* of a conventional language. Other boxes are unnamed,

which makes them rather like *anonymous procedures* or *blocks*. Finally, as we hinted above, the nesting of boxes has a meaning too. For instance, the box named **shape** is defined inside the sprite object **mickey** which means that is **mickey**'s private procedure. Nobody else can use it, except indirectly by **tell**ing mickey what to do.

From our present point of view however, the most interesting thing about Boxer is the way it implements sprites. The first thing to note is that Boxer sprites are also geometry turtles at the same time. In fact the sprites **mickey** and **minnie** in figure 1.10 have a **SPEED** equal to zero and are *only* being used as geometry turtles.

The next feature to notice is the ease with which it is possible to define the sprites' *static* behaviour. For instance, **mickey** has a private procedure which describes what he will do when asked to take a **step**. The other sprite, **minnie**, has a different set of actions for the same command. Also, **mickey** has had his shape redefined to a five pointed star, using exactly the same kinds of commands - Turtle graphics - that are used to make drawings. What Boxer gives us is a particularly neat way of building new objectworlds by allowing us to add new, private, attributes and behaviours to objects.

But the most important innovation is that the sprites can be programmed to respond to *events* - they can be given private procedures which will only execute when a particular action occurs in the Boxer system. For example, in figure 1.10 **minnie** has a (rather simple) box called **m-click** which tells her what to do if somebody clicks the middle mouse button (Boxer systems rely on a three button mouse) over her. There are several kinds of event which sprites can be programmed to respond to - mouse clicks, collisions with other sprites, and collisions with the sides of their box. A second kind of event sensitive processing in Boxer is handled by *triggers* (Klotz, 1989 p10) which are simply boxes that are run when a variable, such as a sprite's speed or heading, is changed.

Together, these features mean that it is possible for a programmer to implement the molecule objectworld used as an example in section 1.12, without disconnecting the user's access to the programming language. Figure 1.11 gives an illustration.

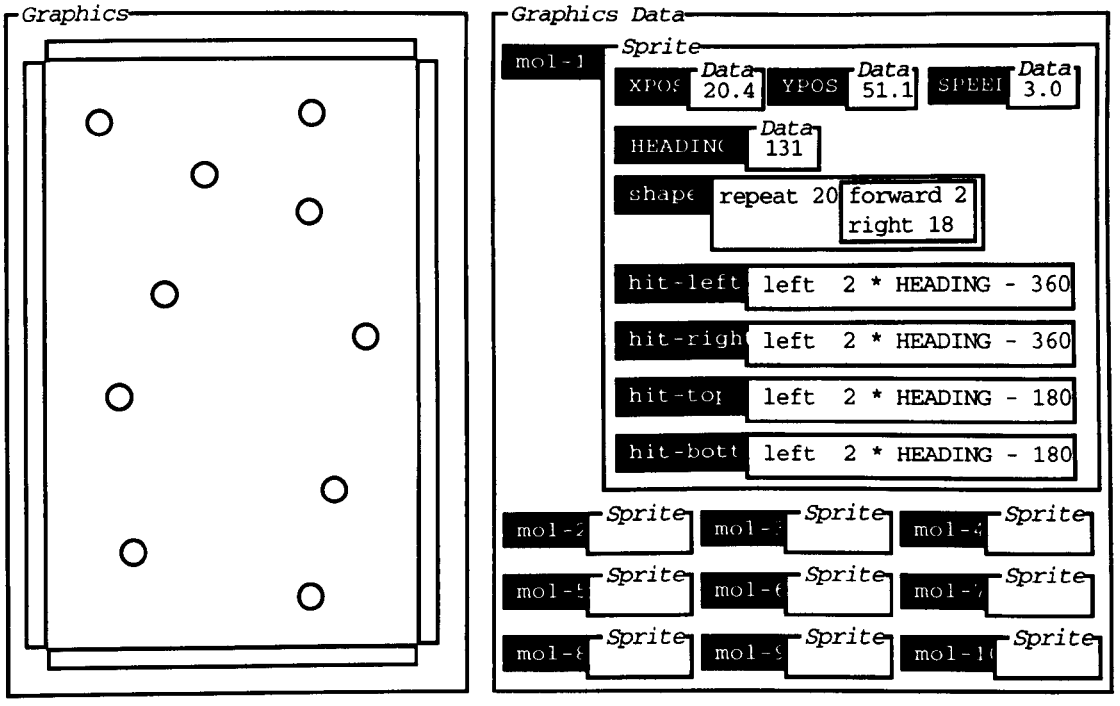


Figure 1.11 A molecule objectworld.

The objectworld shown in figure 1.11 contains fourteen sprites. Ten of these are 'molecule sprites' and the workings of one of them are shown expanded. There are four event processing DOIT boxes, which tell the molecule what to do when it hits the walls. In this simple example there are no collisions between molecules. The walls are actually four more sprites with rectangular shapes whose speeds are set to zero (for simplicity we have omitted their descriptions from the graphics data box).

Now we can see that our hypothetical thermodynamics experiment could be carried out by writing another procedure which operated on the position of the 'wall sprites'. A simulated pressure reading could be taken by arranging for a wall sprite to print out the number of collisions it encounters in a given period.

The significance of these features is that Boxer and its sprites offer designers powerful ways to build objectworlds with new dynamic behaviours, and which preserve the user's access to the programming language. Better still, because Boxer is intended for *naive* users, that is "people who are not programmers but who need to use a computer with more processing ability than word-processing and graphics design" (Klotz, 1989 p10), the designers might be teachers. Members of the Boxer group have carried out research into the ease with which naive users have learned to program the system (diSessa,

1990; Adams, 1989; Ploger and Carlock, 1991). Their findings lend support to the idea that Boxer could open the design of objectworlds up to non experts.

However, at present, Boxer implementations exist only for rather powerful computers. Boxer requires several megabytes of dynamic memory (RAM) to run in, a powerful processor, and a large screen - typically around 1000 by 1000 pixels. Adequate machines are still relatively uncommon even in university departments. No doubt it will be several years before they are common in schools.

1.14 The Logo Culture

There is something artificial about our analysis so far. We are looking back over two or three decades of research and sifting out the things that slot into a narrow category which we have defined with the benefit of hindsight. This suits our purpose, which is to show that objectworlds have a proper heritage, but it leaves out an important human dimension: the culture which grew up around the thousands of teachers and researchers who used Logo and Turtle Geometry in classrooms and laboratories.

Two important themes which ran through the Logo culture in the late 1970s and 1980s were (i) studies performed by educational psychologists to identify specific cognitive changes in students using Logo, and (ii) research into microworlds (where it is to be remembered that this term covers a very wide range of systems). We will deal with each of these themes in turn.

The majority of the cognitive change experiments (e.g. Clements and Gullo, 1984; Pea, Hawkins and Sheingold, 1983; Cathcart, 1990; Swan, 1991) have been concerned with the effects of Logo programming purely on the problem solving abilities of children and have little direct relevance to the school curriculum. However, a group led by Jim Howe at Edinburgh University's Department of Artificial Intelligence was carrying out a broader program of Logo-based research. Howe's team made great efforts to keep their work relevant to the classroom, even going to the trouble of creating their own version of Logo which would run on a microcomputer that was (just) affordable by schools.

Some of the hyperbole surrounding Logo in its early days had remained a source of mild contention in the education community and the Edinburgh group set out to carefully examine the evidence. Citing Feurzeig et al (1969), they identified four main claims:

> "(i) that programming provides some justification for, and illustration of, formal mathematical rigour.
>
> (ii) that programming encourages children to study mathematics through exploratory activity.
>
> (iii) that programming gives insight into certain mathematical concepts, and

(iv) that programming provides a context for problem solving, and a language with which a pupil may describe his own problem solving." (Ross and Howe, 1981)

For claim (i) Ross and Howe conclude there is not much evidence – a study by du Boulay (1978) showed that *some* primary school teachers who were taught Logo programming came to appreciate the value of making explanations explicit. Another study, by Howe, O'Shea and Plane (1979) reported that teachers judged children in a class which had been taught Logo better able to argue about mathematical issues, although this result was clouded by the fact that the experimental group did extra work compared to their peers.

The second claim fares better. Ross and Howe point to abundant evidence for the use of programming as a laboratory for experimental mathematics. One Edinburgh study (Howe, Ross, Johnson, Plane and Inglis, 1982) was carried out over two years and involved two groups of 11 to 13 year old boys. Worksheets were prepared and support software was built (in the form of Logo procedures) for exploratory mathematical activities in the areas of multi-base arithmetic, geometric transforms and algebra. Both the experimental and control groups were tested on their mathematical ability several times during the two years. The researchers conclude that the Logo work had significant benefit, and that the positive effects were especially marked for the less able pupils.

The third claim is also judged by Ross and Howe to have only fairly weak support. They survey four studies which had set out to investigate whether learning Logo had any effects on children's acquisition of mathematical concepts such as *variable* and *function*. Although these studies all reported some improvement for children taught Logo, Ross and Howe question the results, noting that the experimenters were unable to eliminate all the factors other than Logo which may have influenced their performance.

Finally, Ross and Howe look at programming as a context for problem solving. Again, they conclude that the evidence is fairly weak, pointing out that positive findings by Statz (1973) in a study of sixteen 9 to 11 year olds, are flawed by her experimental technique, and an investigation by Papert and Goldstein into children's arithmetic problem solving is of dubious worth because the experimenters neglect the fact that failure in some of the tests they gave children could be due to a lack of background knowledge rather than low

problem solving ability. Overall then, Ross and Howe are equivocal about all but the claim for Logo that it can be an excellent place for children to do exploratory mathematics and investigate the behaviour of mathematical entities.

We now turn to the second theme identified at the beginning of this section. As we have already indicated, the term microworld came to be used very loosely, and during the 1980s this state of affairs continued. For an example take the proceedings of a large Logo conference of the period (Palmgren, 1985). Of twenty authors purporting to describe microworlds more than half are actually talking about systems for which perfectly adequate names already exist. One is clearly a simulation program, written in Logo (Newcombe and Stewart, 1985). Another is a sophisticated text editor and idea processor (Sinclair and Colton, 1985). Still others are about curricula and cognitive states. Many of these papers describe interesting work, but to call them *all* microworlds almost deprives the word of meaning.

This confusion is not necessarily a problem, in fact a workable consensus has evolved around the term to the extent that many Logo workers feel they agree about what they mean when using it. It is worthwhile describing some of the sorts of ideas which contribute to this consensus. As we noted above (in section 1.5), its origins lie partly with Seymour Papert's book *Mindstorms* (Papert, 1980), where he offers several descriptions of his concept, calling them "incubators for knowledge". He characterises them as worlds with their "own set of assumptions and constraints" and as places "where certain kinds of mathematical thinking could hatch and grow with particular ease" (Papert, 1980 p125), but his descriptive explanations give scant assistance to educators who might want to build their own. Furthermore, with a little imagination almost any piece of interactive software could be interpreted as a microworld in Papert's terms.

Fortunately, a few common strands can be discerned in the different things that have been labelled microworld. Typically, a microworld is an open ended program, with little or no internal curriculum, but with opportunities for the user to learn by discovery. One example is the friction microworld developed at the Open University (Spensley et al, 1990) which allows users to explore the motion of bodies over different surfaces. Another common trait is for the microworld software to be open as well, in the sense that its workings

are visible and modifiable. Lawler's function plotting microworld (Lawler, 1982) exemplifies this kind of feature. Neither of these two systems, though, qualifies as an objectworld. The former has no programming language, and the latter no objects. Clearly, objectworlds form only a small part of the microworld spectrum.

However, a vague consensus and a range of disparate examples are not a great deal of help to an educator who hopes to build something comparable in scope to Turtle Geometry. A firmer basis is needed for this task, one built on more precise definitions and a careful analysis of what is special about such systems. That is the aim of this chapter, and by its end we will arrive at a clear definition of what we have come to call objectworlds. Meanwhile, we have already mentioned the attempts by Groen (section 1.11), and Squires and McDougall (section 1.12), to lay the foundations of a theory of microworlds. In the latter half of the 1980s, several other researchers recognised the need for a more analytic approach and we will now consider some of their work.

1.15 *Robert Lawler*

In 1987 Robert Lawler devoted a book chapter to the topic of microworlds (Lawler, 1987 pp1-25). In it he identifies two main features as being the possession of a transitional object (see section 1.6) and the capacity for "creative action" by the user. On the subject of transitional objects Lawler writes:

> "What permits their engaging character is the quasi-concrete instantiations of computational objects which can be taken as symbols by a person." (Lawler, 1987)

That the essential centrepiece of a microworld should be a transitional object is hinted at several times by Papert in *Mindstorms* (Papert, 1980). Unfortunately, Lawler does not give a particularly thorough characterisation of transitional objects or any new examples to complement the Turtle. And while describing creative action he is still rather vague:

> "...the essence is to create an environment in which other people can exercise their own creativity... When the microworld is created with enough structure, it will indicate what objectives and activities are possible." (Lawler, 1987)

Lawler is right to remind us how novel the idea of building a computational environment in which students can be creative is, but his assertion that structure can provide goals and strategies seems too hopeful. If we take Turtle Geometry as a well structured example of a microworld it is still hard to imagine goals and strategies such as those presented in the book *Turtle Geometry* (Abelson and diSessa, 1980) simply dropping into our laps. He goes on to criticise the ubiquitous terminology:

> ""Microworlds" has served as a passable label, but the term does little work for us, because it stands in isolation, unrelated to other ideas in conjunction with which the notions could be better understood." (Lawler, 1987)

He might also have added that the label had been stuck onto too many different things. In his attempt to straighten things out Lawler introduces a new word - "Miniworld". A miniworld is a piece of software which instantiates a transitional object and which permits creative action on the part of the learner. For Lawler microworlds then simply become activities within miniworlds. However, we can see now that it was too late to do anything about the usage, and Lawler's new terminology was not taken up.

A useful distinction made in Lawler's 1987 chapter is his use of the term "microview" to describe the localised cognitive structures or *schemata* that learners build for parts of a problem domain. Some authors of the period were still using the word microworld to describe such structures as well as pieces of software. In fact, the 1987 chapter is not where he first introduced the new term. In (Lawler, 1985) he gives an extended account of a child building up such microviews and uses a chapter to sketch a kind of cognitive theory based on them (Lawler, 1985 p193-209).

1.16 Thompson's Mathematical Microworlds

At around the same time another educational researcher, Patrick Thompson, prompted by his interest in conceptual development and mathematical problem solving, began investigating the possibilities of microworlds. However, he found that the use of the term in the literature had been rather arbitrary: he writes - "It is unfortunate that the generic term "microworld" has been used so many different ways"- but instead of trying to reclaim the word he chooses another course - "Rather than attempt to create a new name, I use the qualifier "mathematical" to distinguish systems described here from what have been called microworlds by others" (Thompson, 1987). Thompson's own definition is interesting:

> "I will use "mathematical microworld" to mean a system composed of objects, relationships between objects, and operations that transform objects and relationships. This characterisation is meant to capture the idea of a mathematical system as constructed from primitive terms and propositions, where the full system exists only potentially but includes features that allow students to expand that potential." (Thompson, 1987)

The microworld he goes on to describe is called MOTIONS and deals with the domain of Transformation Geometry. Specifically, it is intended to be a learning environment for isometric transformations of the plane (transformations which do not change the distances between points). The software is built in a principled way from his definition and his own theory of conceptual development (Thompson, 1985b). The object in the system is the x-y plane, labelled by a small flag. The area containing the flag is continuously visible on the screen and state changes (of position, heading, orientation) are completely described by the flag's appearance. A set of commands to effect these state changes is provided and there is the facility to group and name commands, through a function called DEFINE. By our standards MOTIONS is a fully fledged objectworld.

Thompson describes the objectives behind his system:

> "MOTIONS was designed with a set of cognitive goals in mind. These, briefly, are that students understand motion geometry as a mathematical system, and that they develop concepts of multivariate mappings, invariances under

mappings, and of composition as an operation on mappings." (Thompson, 1987 p86)

Thompson draws on his experience of building and using his system with students as the framework for a methodology of mathematical microworld design. He elucidates some interesting principles: For instance he emphasises that a microworld should be oriented around functions and stresses that there should be "a clear correspondence between the *change* in the display effected by a command and the mathematical meaning of the command." (Thompson, 1987). However, his methodology is somewhat tailored to the domain, transformation geometry, and it would be difficult to generalise it to other areas of the curriculum. The simplicity of the two dimensional x-y plane leads him to undervalue the central importance of the object, its representation and its depiction.

Also in his 1987 paper, Thompson describes some 'problem sets' he found it useful to introduce in classes and shows how the need for such sets affected the overall design of Motions. Thompson examines some of the problems students had when using his mathematical microworlds, such as a failure to explore them in a meaningful way or an inability to generalise some of the ideas they contain. These are, of course, two of the traditional problems associated with discovery learning (see for instance Shulman and Keislar, 1966, chapter 15). Thompson suggests lines of attack on these difficulties and in particular he looks forward to the addition of an artificially intelligent tutor component to the system which would attempt to analyse what the user is doing and offer guidance. He warns that such a component should preserve the passive, non judgmental nature of microworlds by allowing the student to decide when the tutor is invoked.

1.17 LEGO/Logo

In the latter half of the 1980s, in what might appear to be a step back to the early days of Logo, researchers at MIT began connecting computers to *real* objects again. This time, however, the objects were not single purpose robots like turtles but flexible, modifiable machines constructed from special battery-powered electronic LEGO bricks:

> "LEGO/Logo combines LEGO building materials and the Logo programming language. Children begin by building machines out of LEGO pieces - including not only the familiar LEGO blocks, but also LEGO gears, motors, and sensors. Then children write Logo computer programs to control the machines that they have built." (Martin and Resnick, 1990)

Unlike the Turtle, these new robots do not have to be connected to a computer by wires. Instead, the computer, in the form of a special Logo *programmable* brick, can be incorporated into the machine. The programmable brick is essentially a miniaturised Apple II computer running a version of Logo with extensions which allow it to control external devices. There are three other types of Logo brick: *Actuator* bricks, such as motors, lights, and bleepers, are the components which "create interactions with the outside world" (Martin and Resnick, 1990). *Sensor* bricks detect features of the environment, such as sound, light, and touch. Finally, *logic* bricks allow the outputs of sensors to be combined and processed before they are passed to actuators or a programmable brick. For example, the designer may want a machine to respond only when both sound and light are present. To do this the outputs of two sensors may be passed through an *AND* brick, and then to an actuator brick. Simple machines can be built using just actuator, sensor, and logic bricks, but the addition of the programmable brick means much more complex behaviours may be investigated.

The researchers who developed LEGO/Logo describe it as a *Scienceland* for the classroom, and they claim that it can make school science more relevant for students:

> "By working on LEGO/Logo projects, children deal with scientific concepts and methods in a natural and meaningful context. Children don't just learn about science. They do science." (Resnick and Ocko, 1990)

Learners building LEGO/Logo devices have to make many different kinds of decisions about overall plans, choice of components, and programming the bricks. Resnick and Ocko characterise the typical activities of children using the system as learning *through* design and learning *about* design.

In our established terms, LEGO/Logo seems to be a new objectworld. After all, they have objects and a language with which to manipulate them. However, there is an important difference between LEGO/Logo and the systems we have been discussing. The fact is that LEGO/Logo machines are real, and a whole world of constraints unavoidably act on them. Motors and gears have friction, wheels slip, and batteries run down. Because of this LEGO/Logo is less suitable for engineering the *transitional* objects of section 1.6. An important feature of transitional objects is that they are part concrete and part abstraction, and they can therefore help to form a link between what the learner already knows and the formal concepts they have to acquire in school. As Papert puts it :"...an entirely new kind of object - a transitional object between the ones that you can touch and push (like tables and wooden blocks) and the kind of objects that you know in science, in philosophy, and in mathematics" (Papert, 1987a p88). Of course, a child who has built some machine in LEGO/Logo may internalise the experience and use it as a transitional object later on. This is like the experience with gears Papert describes in the foreword to *Mindstorms*.

It is important to recognise that this is *not* a negative criticism of LEGO/Logo. In fact the real world considerations, which are missing from an objectworld such as Turtle Geometry, are valuable parts of the LEGO/Logo experience. It simply puts the LEGO/Logo approach into a different, complementary category, which is appropriate to different kinds of learning.

1.18 Definition

We now present a definition of objectworlds, which is intended to be of use to designers who would like to build a system of this class for their own domain of interest. The definition is part of our wider goal, mentioned in the preface, of encouraging the construction of many more objectworlds. In turn, this goal derives from one of Papert's beliefs about computers in education, that for their effect to be felt, there is the need for a multitude of such systems, connected together to form a curriculum (Papert 1987a). At first sight the definition may seem rather abstract so we will follow it with some explanatory notes.

> A computer based objectworld is the combination of a simulated object (or objects) and an interactive programming language. The object should be continuously visible and its attributes should derive from, or be a representation of, some fundamental concept. The language should contain a set of commands that allow the inspection and manipulation of the object's attributes, and must support data types corresponding to those attributes. At least some of these commands should act on the objects in ways that we would expect learners to grasp with little difficulty.

1.19 Notes on the definition

(i) As we have said, the pairing of a simulated object and a programming language was inspired by Turtle Geometry. The constraint that the objects should be continuously visible is a feature of Turtle Geometry too, but the idea also draws support from the field of Human Computer Interaction (HCI). For instance, Shneiderman emphasises its importance for *Direct Manipulation* (DM) interfaces:

> "The object is displayed so that actions are directly in the high-level problem domain. There is little need for decomposition into multiple commands with a complex syntactic form. On the contrary each command produces a comprehensible action in the problem domain that is immediately visible." (Shneiderman, 1983 p66)

(ii) The "fundamental concept" on which the object is based obviously defines the knowledge domain to which the objectworld will apply, but there is an implicit limitation to the objectworld approach here: while many basic scientific and mathematical ideas seem easy to represent on a computer, it is not difficult to think of things, such as the concept of irony, which might be more problematic.

(iii) The language we have in mind is Logo or something of equivalent power. That is, it should be possible to build independent procedures, and it should support structured data types. For instance, a position is usually represented by two numbers. Logo *lists* allow us to define a single variable which holds two numbers.

(iv) There needs to be a pair of commands for each attribute of the objects - one to set and one to inspect the values. For example the Turtle has **setxpos** *number*, which sets the x position, and **xpos** , which returns it. In addition, the Turtle has commands that are easier for children to understand, **forward**, **back**, **right**, and **left**. Papert calls these *body syntonic* commands (Papert, 1980 p63) because they connect to knowledge that the learner already has about moving and turning.

1.20 *Summary*

We have described an evolving family of computer-based discovery learning environments which began with the invention of Turtle Geometry in the early 1970s. The family has relatively few members compared with, say, the number of educational simulations that are in existence. However, this class of environments is distinct enough to warrant its own name, and we have chosen objectworld.

Turtle Geometry, and therefore objectworlds, arose out of the development of *conversational* programming languages in the late 1960s. These systems allowed users to see the results of computations immediately, without the delays of compilation and batch processing. The genesis of objectworlds was the decision of Wallace Feurzeig and Seymour Papert to design a conversational language specifically for children and link it to a small object, the Turtle, capable of moving and turning.

Impressed by the success of Turtle Geometry, Papert formed a group of Logo researchers at MIT. Investigations into the use of Turtle Geometry by children were carried out, and new objectworlds, such as those based on biology and physics turtles, were created. We indicated that in some cases these new turtles had time-dependent *dynamic* behaviours, generated by simple recursive Logo procedures. Since most versions of Logo are capable of running only one procedure at a time these dynamic objects came at the expense of *continuous* access to the programming language. We call systems such as these *restricted* objectworlds.

In the late 1970s, Papert began to develop ideas which led to a firmer psychological basis for objectworlds. In particular he began to view entities like the Turtle as *transitional objects* which, when equipped with *syntonic commands* to control them, can serve as stepping stones between a child's personal, intuitive knowledge and the formal concepts we expect her to learn. The Turtle is a transitional object because it turns and moves in ways children readily identify with, and yet it also connects to higher level mathematical concepts. The degree to which this simple "cybernetic animal", the Turtle, can be a vehicle for profound mathematical investigations is illustrated by the contents of the book *Turtle Geometry* (Abelson and diSessa, 1980).

diSessa and White used objectworlds based on *Dynaturtles* to investigate children's ideas about motion. In particular, they were able to get learners to reconsider some of their incorrect intuitions, such as the "Aristotelian" belief that an object will move in the direction it was last pushed.

We have shown that another evolutionary step was made with the introduction of *sprites* to some versions of Logo in the early 1980s. Sprites are *dynamic* objects but they do not interrupt the opportunity for learners to write programs. A separate, background process keeps them moving smoothly across the screen at a velocity that is under the control of the user. We have argued that for reasons of clarity and understandability, it is desirable for dynamic behaviour procedures to be run in *parallel* with the general processing ability.

We have also shown that during the 1980s several researchers voiced their concern at the indiscriminate way in which the term microworld was being applied to different learning environments. In particular Groen (1984) and Squires and McDougall (1986) present definitions of the term which incorporate what they see as the essential features of these systems. Groen emphasises the need for reversible operations in microworlds, while Squires and McDougall concentrate on the combination of operators (acting on the state of the object) and a programming language. Thompson (1987) also notes the confusion surrounding the microworld label and adds the qualifier "mathematical" to it for discussions of MOTIONS, the objectworld he has built for the domain of Transformation Geometry. The definition we presented in the last section has been influenced by these and other authors.

The next stage in the development of objectworlds was made with Boxer (diSessa and Abelson, 1986) which offered users a range of ways to create new objects with independent attributes and behaviours. These are set in an innovative programming environment which extends the familiar "What You See Is What You Get" notion to the description of computational processes. Furthermore, Boxer has a mechanism which allows the definition of new *dynamic* behaviours for sprites, without interrupting access to general processing.

The historical survey outlined above is intended to provide a context for Gravitas, the objectworld at the centre of this thesis. The next chapter contains an in-depth description of its design, construction, and capabilities.

2 Gravitas

2.1 Overview

Towards the end of the last chapter we presented a definition for an evolving class of educational computing systems which we call *objectworlds*. Our practical working out of this definition (which builds on the notes of section 1.19) may be taken as follows:

To build a new objectworld, think of an entity which embodies the concept (or concepts) you wish to let students explore. Next, devise a simulation of the entity (in our terms this means a representation of its attributes, the algorithms which generate its behaviour and a depiction to make it visible on the screen) which can be executed on a computer. Now connect the object to a programming language with a set of commands which allow each of its attributes to be set or inspected. Finally, add another set of commands which act on the object in intuitive ways. That is, in ways it is reasonable to expect the learners to comprehend immediately.

In this chapter we will show how the above formula led to the design of Gravitas, a new objectworld for physics education which also introduces several innovations. Gravitas allows learners to build systems of gravitating masses in a two-dimensional space and observe their dynamics. Users can create as many objects as they like and may freely manipulate their attributes, which are position, velocity, mass and radius. Gravitas displays the objects in a space whose size may also be defined by the user. Gravitas is attached to a Logo interpreter so that learners may write programs which manipulate the objects and the space.

The central objects of Gravitas are called Massobs and, like Turtles, they can be controlled and inspected by a set of Logo commands. Among their many commands Turtles always have four special ones, called **`forward`**, **`back`**, **`right`** and **`left`**, which move them around the screen but which are especially easy for novices to appreciate because they do not require any understanding of coordinate systems or angles. In the same way, Massobs have four simple commands which affect the *rate* at which they move around the screen but which do not use coordinates or angles. We will describe the origins of these and the other Massob commands in the sections which follow.

From the beginning we decided to make Gravitas similar, in both its overall structure and its modes of use, to Turtle Geometry. We believed it would help new users if they could relate Gravitas to something they have already learned, and Turtle Geometry has been a very popular system. This chapter will attempt to make the similarities explicit.

We will also cover a major difference between Gravitas and Turtle Geometry. Massobs are more complex objects than Turtles, in the sense that they have more attributes, and therefore they are more difficult for young learners to appreciate. To alleviate this problem, we have given Gravitas a mouse-driven graphical interface which is quick and easy to use, compared to typing in commands. This interface is described towards the end of the chapter.

2.2 *General Description of Gravitas*

The overall design of Gravitas is clearly similar to Turtle Geometry, with its basic pairing of objects and Logo. Just like that system, the objects of interest may be controlled by simple commands, and the commands can be assembled into procedures which encapsulate more complex effects. However, it differs from Turtle Geometry in several important respects. First of all, the central objects of Gravitas, which we call Massobs, are *dynamic*: they move continuously across the screen leaving a visible trace. Secondly, the trajectory of each Massob is affected by the presence of others: they interact through the force of gravity. Thirdly, Gravitas allows users to construct as many Massobs as they wish. Most versions of Logo restrict users to one or just a few Turtles.

To begin our description of Gravitas we will look at a typical session. The user was interested to see how two objects, in similar orbits around a third, would interact.

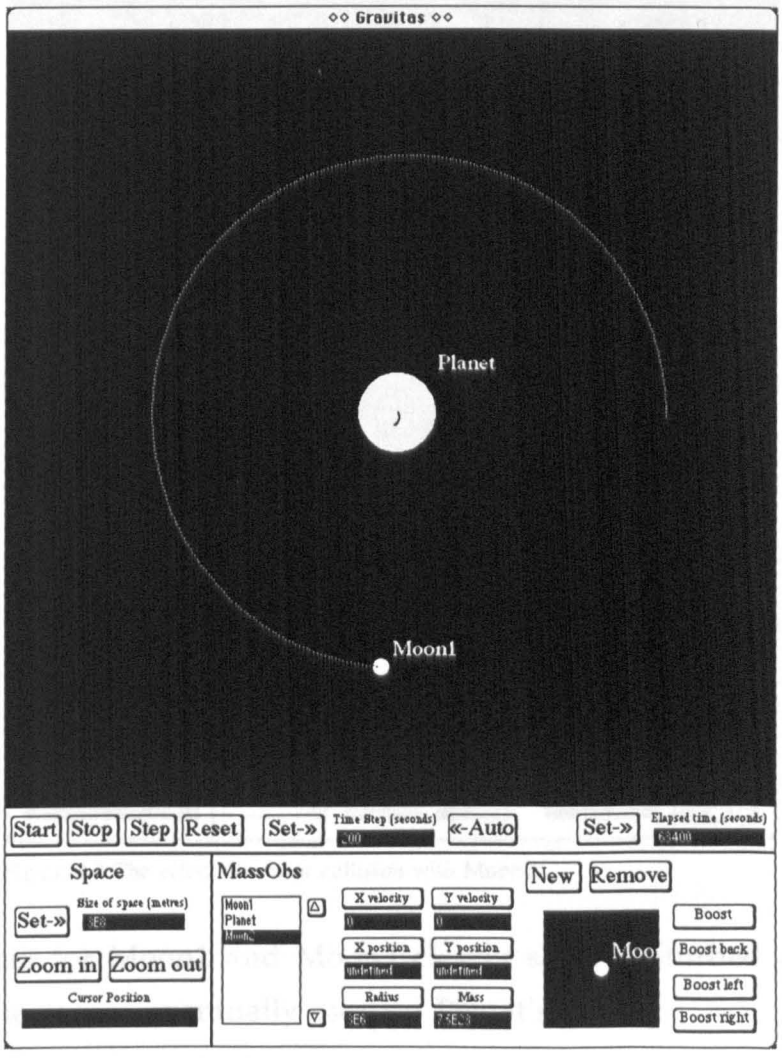


Figure 2.1 Gravitas showing two active Massobs, with a third under construction

In figure 2.1 a user has already built two Massobs, Planet and Moon1, and they can be seen orbiting one another. A third object, Moon2, is being manufactured in a small window at the bottom right of the screen, and is ready to be dragged into the space where it will interact with the others. The dark area of the window they inhabit represents a space $3x10^8$ metres across. This size, roughly the distance travelled by light in one second, was defined by the user, who has also set the size and the velocities of the individual Massobs. The Planet is roughly two and a half times the diameter of the Earth.

In figure 2.2 the user has dragged Moon2 out of the "factory" and into the space, to a position quite close to Moon1. Of course, if this really happened it would be a cataclysmic event, but with Gravitas, it is easy to watch the consequences.

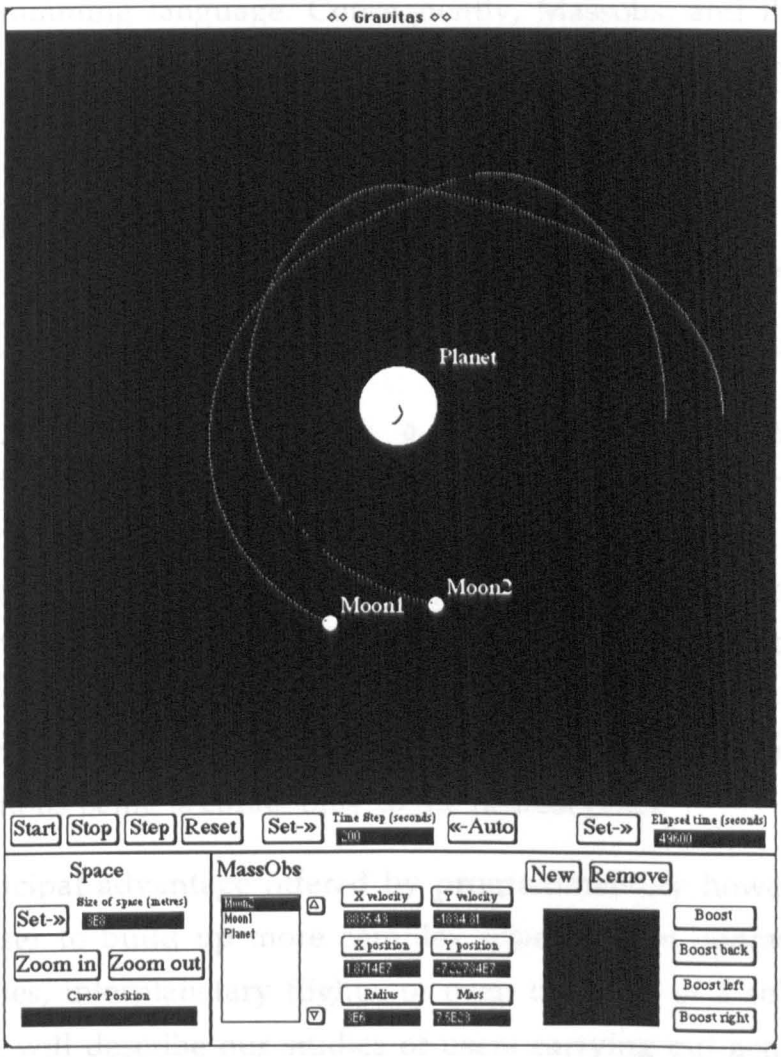


Figure 2.2 The effect of a near collision with Moon2.

The traces for Moon1 and Moon2 clearly show disturbed orbits, and in fact, one of the moons eventually escaped Planet's gravity.

The session was carried out using Gravitas' sophisticated *graphical* interface, which allows users to begin constructing and exploring such systems after just a short period of familiarisation. Using mouse clicks and drags learners can create Massobs in the factory and set their masses, sizes and velocities. Gravitas automatically pauses the simulation while Massobs are dragged out into space, so that they may be positioned accurately with respect to others. Once they are in space the Massobs obey Newton's laws of gravity and motion, tracing out their trajectories on the screen. The simulation may also be turned on or off by clicking on buttons at the bottom of the window.

However, the graphical interface is not the only means of operating the system. Gravitas is an objectworld, and from our definition that means it must have a programming language. Consequently, Massobs, and the space they exist in, may be controlled by *programs*. Gravitas contains a version of Logo which has been extended with commands dealing with these new entities. For example, the session described above could be generated by a few lines of code, as shown in figure 2.3.

```
create.space  3.0E8
create.massob "Planet  0        0  0  0      1.5E7  8E25
create.massob "Moon1    1E8     0  0  7500   3E6    7.5E23
go.until.time  63400
reset
create.massob "Moon2    1.2E8   0  0  5000   3E6    7.5E23
go.until.time  49600
```

Figure 2.3 Logo code for the session represented by figures 2.1 and 2.2.

As we can see from figure 2.3, using the *programming* interface means that the learner must supply numeric values for sizes and positions which the graphical interface allows simple mouse operations to control. On the other hand, the programming interface offers repeatability and greater precision, mouse operations being accurate only to the nearest pixel.

The principal advantage offered by programmability however, is that it allows the user to build up more complex *sequences* of operations, such as rocket launches, interplanetary flights, or even the birth of a solar system. In chapter 3 we will describe our studies of users carrying out activities such as these. Programmability also makes possible the construction of *extensions* to Gravitas, for example a function to measure the kinetic energy of a system of Massobs, in Logo, a standard and popular programming language. In chapter 4 we will present some example extensions.

The overall structure of Gravitas, therefore, parallels Turtle Geometry in that the objects of interest may be controlled by programs. A deeper similarity between the two is the provision in Gravitas of simple to understand commands which newcomers to the system can use to manipulate Massobs. Novice users of Turtle Geometry begin by using coordinate independent commands like `forward 100` and `right 90` to control the Turtle. These are often called the Turtle's *body syntonic* commands (Papert, 1980 p206) because they can be understood in terms of sensori-motor knowledge about moving and turning that even young learners are bound to have, and they do not require an appreciation of concepts like coordinates and angle. Similarly, Gravitas supplies four body syntonic commands for use with Massobs. These commands - `boost`, `boost.back`, `boost.right` and `boost.left`, accelerate Massobs forwards, backwards, to the right, or to the left, without requiring the learner to think about the Massob's current velocity, or position, or mass. We will discuss the boost commands in more detail in section 2.2.3.

Gravitas runs on Apple Macintosh II computers alongside the Logo interpreter which may be used to program it. The number of Massobs which users may create is limited only by available memory, but the amount of computation which Gravitas has to perform rises in proportion to the *square* of the number of objects defined. Inevitably, there comes a point when the system response slows below the level of acceptability.

In the next section we will describe Massobs in some detail: how they are implemented, how their behaviour is generated, and how they may be programmed. Then we will move on to discuss the two interfaces: first the programming interface, and then the user-friendly graphical interface.

2.2 *Massobs*

Throughout this thesis we have defined Turtle Geometry as the realm of activities that are opened up by the combination of a programming language, usually Logo, and an object, the Turtle, which can move and rotate and leave a trace of its movements on the screen. Similarly, Gravitas gives access to a new range of activities, which we call Massob Physics, through its joining of Logo and Massobs. It is therefore natural to consider Massobs as counterparts to the Turtle, and throughout this section we will draw parallels between them to illustrate important points.

2.2.1 *Attributes and Values*

Turtles and Massobs both have a state. For the Turtle this is represented by three attributes: a *position*, represented by two numbers, a *heading*, expressed as an angle, and a *pen*, which may be up (in which case the Turtle leaves no trace as it is moved) or down. A Massob's state, on the other hand, is composed of five attributes: *position*, *velocity* (also stored as a number pair), *mass* and *radius*, and finally a *name*. Like Turtles, Massobs leave a trace on the screen as they move.

Massobs follow Turtles in that they attempt to make the values of these attributes apparent from their appearance on the screen. For instance, a Turtle's depiction, commonly as a small isosceles triangle, makes its attributes immediately visible: position is obvious, the heading is roughly indicated by where the triangle seems to be pointing, and a dot at the base of the triangle indicates whether the pen is up or down. The same is almost true of Massobs. Looking at figures 2.1 and 2.2 it can be seen that their positions and names are obvious, and from the traces (or simply by observing them for a few moments) we can see the directions in which they are moving. Their relative sizes are also easy to see. The exception is mass. If all the Massobs in a system have the same density then their relative masses can be inferred from their size (actually, from the *cube* of their size). However, the user is free to alter the density of Massobs by independently varying their mass and size, which means that a small Massob does not necessarily have less mass than a large one.

Of course, the values of attributes cannot be read with much accuracy from their appearance on the screen, either for Turtles or Massobs. However, Turtle Geometry has a pair of commands for each attribute - one to inspect and

one to set the value. Gravitas has an analogous set of paired commands to read and set the *exact* values of Massob attributes. These commands, which take the form of extensions to Logo, form part of what we call the *programming interface* to Gravitas, which we will be describing in section 2.4. For example, the command **x.pos** takes a Massob as its input and returns the x-component of its position. Thus typing **print x.pos :moon1** into the Logo interpreter running alongside figure 2.2 would produce the result **1.53906E7** (i.e. 1.53906×10^7).

Additionally, Gravitas has another separate but completely equivalent means of setting and inspecting Massob attributes, which we call the *graphical interface*. This feature allows the user to set and inspect the precise values of the attributes using only the mouse. The graphical interface will be described in detail in section 2.5.

2.2.2 *The Computational Nature of Massobs*

Before proceeding, we should clarify a rather technical detail concerning the *computational* nature of Massobs. So far we have discussed Massobs in an informal way. We introduced them by their *names*, such as Moon1, then showed that they could be created with a command such as **create.massob "Moon1 1E8 0 0 7500 3E6 7.5E23**. Subsequently, Moon1 was referred to in a procedure call as **:moon1**. There is some scope here for confusion.

First of all, when we create a *variable* in Logo, we are associating a symbol with a value. Thus **make "foo 3** associates the symbol foo with the value 3. We can retrieve the value placed in **"foo** by typing **thing "foo**, which is usually shortened to **:foo**.

Similarly, **create.massob "Moon1 1E8 0 0 7500 3E6 7.5E23** creates a variable, **Moon1**, whose value is a new Massob, and it sets the attributes of the new Massob, in the order name, x and y position, x and y velocity, radius and mass. This Massob can be retrieved in the same way as any other value by typing **:Moon1**, and since Logo is not concerned about case in variable names **:moon1** works just as well. It is important to note that the name of the variable and the name displayed by the Massob on the screen are the same. This is a great convenience as it is natural to refer to a Massob in a program by the same name it displays on the screen. However, it does make it inadvisable to include space characters in names as they make the variables difficult to use.

But what are Massobs in a computational sense? They are not simple numeric values, like 3 or 5.748, nor are they Logo lists of data such as `[Moon1 1E8 0 3E6 7.5E23 0 7500]`. In fact, they are *objects* in the sense defined by the discipline of Object-oriented Programming (Goldberg and Robson, 1983; Cox, 1986; Drescher, 1987; Graham, 1991). In this paradigm an object is usually described as a collection of private *instance variables* and *methods*. Thus a Massob like Moon1 is composed of instance variables, containing its attributes as private data, and methods which are its personal copy of the procedures that allow it to *do* things, such as display itself, gravitate with other Massobs, and leave a trace on the screen.

Furthermore, Gravitas was not only built using an Object-oriented style of programming; the Logo interpreter which runs alongside it is also Object-oriented. The result is that users may, if they wish, program the system in an Object-oriented way. For most users of Gravitas this feature will not be significant, since only those with relatively strong programming skills will be able to exploit it. However, we considered it to be important for the future of the system, as it opens up a very clear path for future development: Gravitas is a prototype-based Object-oriented system rather than a class-based one (Drescher, 1987) and Massobs can be used as prototypes for new, more complex objects which inherit all their ancestors' properties, and then add some of their own. It is possible to envisage descendants of Massobs which have spin, or a different appearance, or a magnetic field.

2.2.3 *The Boost Commands*

As we mentioned in section 2.2.1, Turtles and Massobs have commands which set their attributes *directly*. For instance, `setposition [50 50]` moves a Turtle towards the top and right of the screen, while `set.velocity [10 10]` sets a Massob moving in the same direction. However, these are not normally the commands to which learners are first introduced. Novices usually begin Turtle Geometry by using the *coordinate independent* commands `forward`, `back`, `right` and `left`. In view of this, at an early stage in the development of Gravitas, we decided to provide Massobs with coordinate independent commands for controlling their velocity. The commands we developed are named `boost`, `boost.back`, `boost.right` and `boost.left`.

The boost commands accelerate Massobs forwards, backwards, to the right or to the left, *with respect to the direction the Massob is already travelling in.* They take a single input - the name of the Massob to be boosted. The size of the velocity change they produce is set with another command, called **`set.boost.strength`**. These commands are so important to Gravitas that we should examine them in some detail.

Coordinate independent commands do not act directly on the position attributes of an object. Rather, they use intermediate procedures to modify them. For the conventional Turtle these procedures are quite simple. The conversion of the Turtle's heading is straightforward: the commands **`right`** and **`left`** simply add or subtract their arguments from it, under the constraint that it is kept in the range 0 to 360 degrees. Figure 2.4 shows how **`forward`** and **`right`** are turned into a change of the Turtle's position and heading by some basic trigonometry.

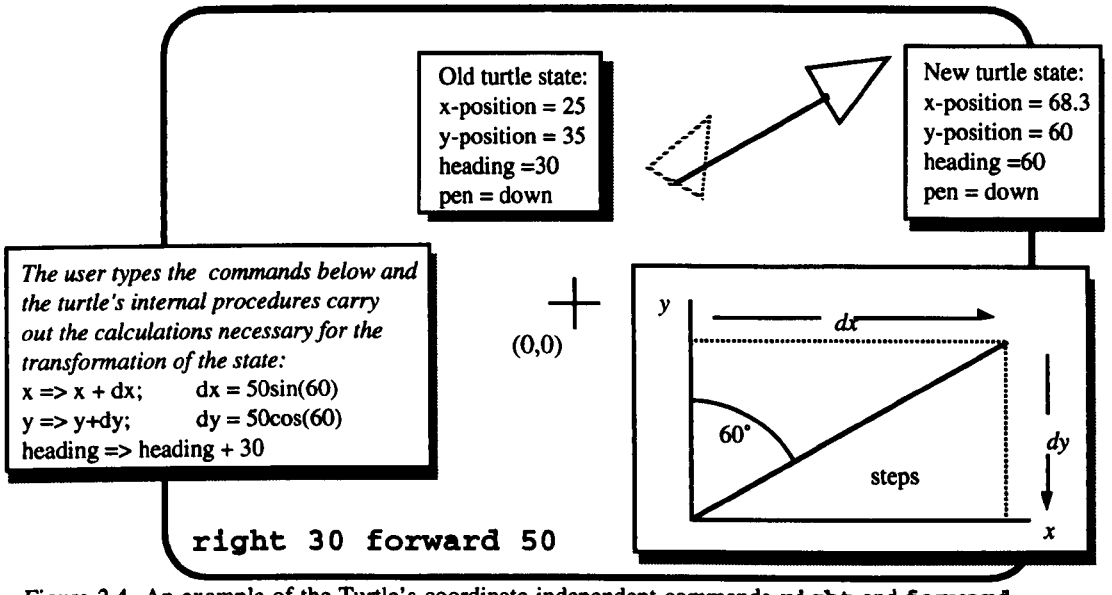


Figure 2.4 An example of the Turtle's coordinate independent commands **`right`** and **`forward`**.

The boost commands operate like 'kicks' which instantaneously change a Massob's velocity in one of four directions, as illustrated in figure 2.5. The amount by which the velocity is changed does not depend on mass - for a given boost strength (which the user may vary) the increment is the same for any Massob. Also, the velocity changes take place instantaneously. For this reason it is better to describe the effects of the boost commands as velocity increments rather than accelerations.

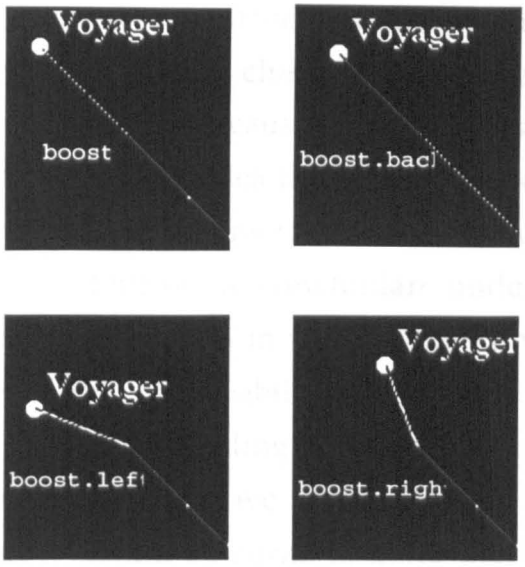


Figure 2.5 The effect of the four boost commands on a Massob

Papert describes Turtle Geometry's **forward, back, right** and **left** commands as being *body syntonic*. The term is given some prominence in chapter 3 of his book *Mindstorms* (Papert, 1980). According to Papert, syntonic learning takes place when the knowledge to be communicated is firmly related to intuitive knowledge the learner already possesses. Thus **forward, back, right** and **left** are body syntonic because we already know what it is like to move and turn. This contrasts with the commands that directly affect the Turtle's attributes, which require that the user already understands the formal concepts of coordinate systems and angles.

Gravitas' commands **boost, boost.back, boost.right** and **boost.left** are syntonic on one level, in that everyone has sensori-motor knowledge of what it is like to be pushed around. But they are also problematic, in that human intuitions about pushing are formed in an environment which contains friction and where gravity always acts downwards, whereas Massobs exist in a frictionless space where the force of gravity acts between mass centres. This is a very important point, because it shows that there is a crucial difference in understandability between Turtle Geometry and Gravitas: novices usually have no trouble with the idea of a Turtle command like **forward 100**, nor are they surprised by its effect - a line on the screen 100 units long. However, although the ideas of the **boost** commands are also grasped without difficulty, their *effects* - Massobs which continue moving indefinitely, and perhaps at unexpected angles - come into conflict with some common misconceptions children hold about motion.

These are the same misunderstandings noted by diSessa in his work with Dynaturtles. As we noted in chapter 1, he calls them Aristotelian preconceptions (diSessa, 1982), because they accord with some of Aristotle's beliefs about motion, such as the idea that a body moves in the direction it was last pushed. This equivalence means that Gravitas could, like Dynaturtles, be used to help learners develop a Newtonian understanding of dynamics although our studies of the system in use (which we will describe in the next chapter) investigated different possibilities for Massobs. However, because of this possibility for misunderstanding it is more accurate to call the boost commands semi-syntonic. In fact, we will stick with the shorter form in the rest of this thesis but it should be borne in mind that the boosts do carry this extra complexity.

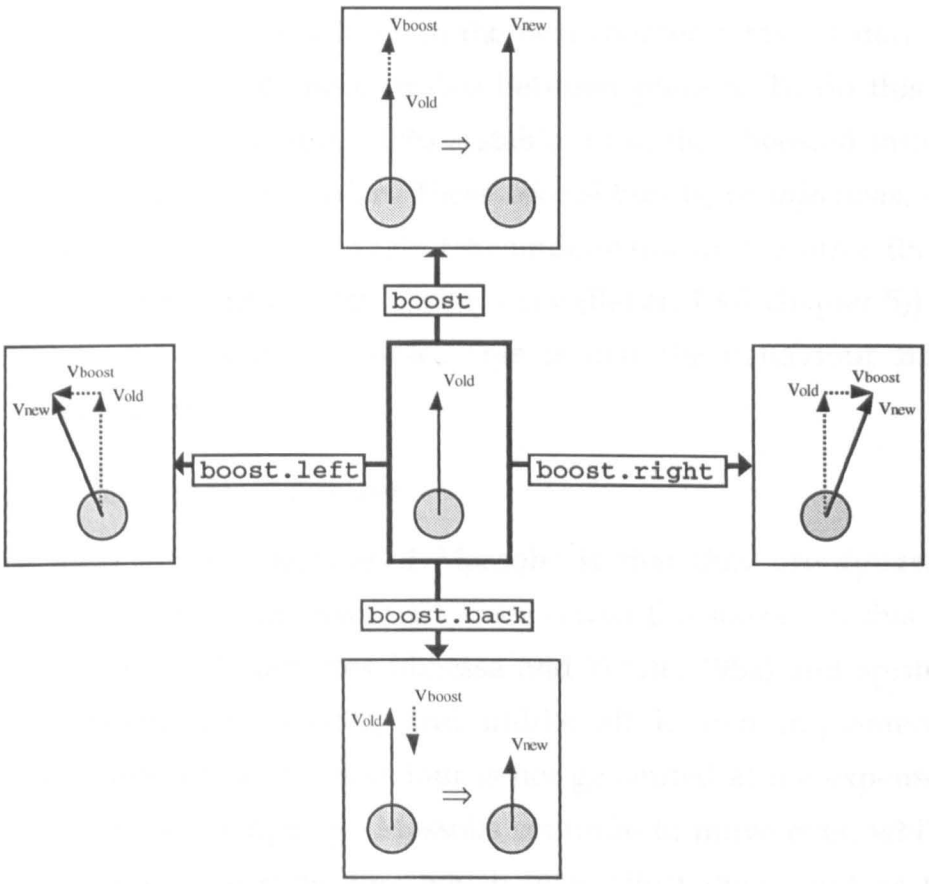


Figure 2.6 How the four boost commands alter the velocity of a Massob

Figure 2.6 shows the detailed effect of the boost commands as they are implemented in the version of Gravitas used in our formal studies. It should be understood that this is not the only way we could have implemented syntonic commands for Massobs. In fact, we tried other schemes in different versions of Gravitas, and we will describe some alternatives in section 2.8. An important decision was to make the boosts act as velocity rather than momentum increments. This was done because masses can range over many

orders of magnitude and therefore an appropriate momentum boost for one object might be negligible or unreasonably large for another. Behind the scenes, Gravitas actually computes a "reasonable" default value for the velocity increment used by the boosts, where reasonable means a visible change of one or two pixels per frame of the animation.

However, there is another reason for the current behaviour of the boosts. Early informal studies of Gravitas with a number of users had already pointed to a line of investigation for the formal studies which we decided to carry out. Consequently, we made sure that the boost commands installed in Gravitas were well suited to the class of tasks we envisaged: experiments in orbital mechanics. In particular, it is very useful to be able to apply a boost in exactly the direction a Massob is travelling, even if its trajectory is being curved by the presence of others. As we will see in the next chapter, Gravitas may be used to simulate the journeys of space probes between planets. To do this the probe must be launched and boosted into a stable orbit, then boosted into a transfer orbit toward its target and so on. These orbital boosts, or *injections*, as they are called in astrodynamics, are easiest to understand and control (both from a theoretical and a practical point of view - see (Baker, 1967 chapter 5)) if they are tangential to the vehicle's course. This is just the behaviour **`boost`** and **`boost.back`** provide.

2.2.4 *Dynamic Behaviour*

The most striking feature of Massobs is that they are *dynamic* objects which move continuously, leaving a trace across the screen. In this sense they are descendants of Dynaturtles (diSessa and White, 1982) and sprites (Papert, 1987a). However, like sprites, and unlike all known implementations of Dynaturtles, their dynamic behaviour is not generated at the expense of access to the programming language. Massobs continue to move even while the user writes and runs Logo programs which may affect them, and as they move Gravitas keeps track of the elapsed time. This contrasts with the conventional Turtle which remains stationary between commands and for which time has no meaning. Gravitas makes sure that Logo is always available for use, in a separate window, and the language has been extended with all the commands needed to create, control and destroy Massobs.

The second aspect of their dynamic behaviour is that Massobs interact with each other gravitationally. It is this feature which clearly sets them apart

from sprites. Figure 2.7 shows Io and Europa moving around Jupiter leaving a trace of their orbits. They were not specifically programmed to orbit Jupiter, but were created with the appropriate initial conditions (a certain distance from Jupiter and a tangential velocity) and they are obeying the law of gravity. All Massobs 'know' how to feel the gravitational attraction of all other Massobs, in the sense that they have a method which can compute it. In general there are no workable analytic solutions to the equations of motion for systems composed of three or more bodies, so we have equipped Massobs with a numerical method which they use to compute their trajectories. The technique we have employed comes from Celestial Mechanics, and is known as the Method of Special Perturbations (Roy, 1978). Briefly, this involves integrating the equations of motion over a suitably short time increment, updating the position and velocity, and then repeating the process for as long as desired.

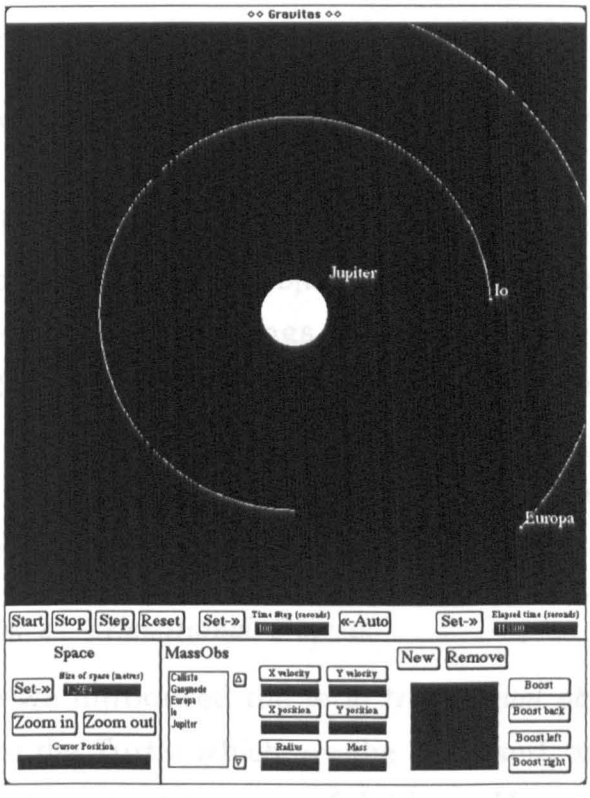


Figure 2.7 Io and Europa orbiting Jupiter

Like most numerical techniques, the Method of Special Perturbations is subject to inherent inaccuracies. In effect, this is the price we pay for obtaining any answer at all to a question - the so called *n-body problem* (Roy, 1978 chapter 5) - for which there are no general solutions. These errors are scarcely a problem in situations where the velocity increments of Massobs are modest. For instance, the error is insignificant in orbits of low eccentricity, where 'low' would include all the orbits of the planets in the solar system. Furthermore,

when the orbits the user is interested in do become eccentric or highly curved, the size of the errors can be reduced by shortening the time step on which the integrations are based. This may be done by hand but Gravitas can also be asked to perform the task automatically.

Beyond this, there are a number of strategies and heuristics which can be used to minimise the problem of numerical errors. Many of these are to be found in standard texts on Celestial Mechanics and Astrodynamics (e.g. Roy, 1978; Baker, 1967) and in the literature of Astrophysics. We have tried some of them in prototypes of Gravitas but they are not included in the present version because the gains in accuracy were not felt to outweigh the performance penalties of the extra computation. However, this situation will certainly change in the future, with faster computers and more efficient implementations. With this in mind we provide a survey of the relevant techniques in Appendix A.

Two final aspects of Massob dynamic behaviour concern what happens when they move off screen and when they collide. On its default settings Gravitas simply allows Massobs to move off the screen unhindered, but an option may be selected under which objects going off screen trigger a zoom out (see section 2.3 below) which brings them back into view. The default behaviour in a collision is for the Massobs to coalesce to form a new object with the combined mass and momentum of the originals. This behaviour was chosen because it is the most physically realistic: Fragmentation is a rare phenomenon in real collisions of planetary objects.

2.2.5 *Massobs as Transitional Objects*

In chapter one we introduced the term *transitional object*. This is Papert's name for objects like the Turtle which, on the one hand, are easily understood by children, and yet connect to powerful ideas. He points out that many students have trouble in crossing the gulf between their personal, intuitive knowledge about the world and the things in it, and the formal, theoretical objects they are expected to learn about in school. As he puts it:

> "Science is full of stuff like electrons, genes, and quasars. Mathematics is full of the square root of minus one, or even the number 562. These are not really things you can touch. Many children and older students have trouble when they first run across objects like these." (Papert, 1987a, p88)

In Papert's view, transitional objects are rather like stepping stones which bridge the gap between the concrete and the formal. We may help ourselves by building our own transitional objects: in a well known passage in the foreword to *Mindstorms*, Papert describes how his childhood fascination with gears helped him grasp the formalisms of arithmetic. But in Papert's opinion, a profound possibility offered to education by computers is that we can "engineer" transitional objects, like the Turtle, sprites, and Dynaturtles, to suit more tastes and situations.

However, computer engineered transitional objects seem to be thin on the ground, a state of affairs regretted by Robert Lawler in a 1987 book chapter (Lawler, 1987). He offers a few reasons why this may be the case, such as the lack of a sound theory and confusing terminology. What he doesn't say though, is that they might be very difficult to build. The Turtle seems so simple, its workings so straightforward, that we might be excused for overlooking this. However, in the light of our experience it seems likely that the implementation of transitional objects for some concept areas will be a large undertaking. For example, although Massobs appear outwardly simple, they actually have some quite intricate machinery supporting them, and a substantial effort went into their design and construction. Even the ordinary sprites found in some versions of Logo are implemented by far from trivial mechanisms which are often taken for granted.

Massobs can be thought of as transitional objects because although they connect to the formal concepts of massive bodies from Newtonian physics, at the same time they can be manipulated in ways that may be appreciated by children. It is sensible to make the cognitive effort required of a learner working with a transitional object as small as possible. In Gravitas, this effort is reduced because the learner does not have to think too hard about the *direction* of the boosts - they are consistently either co-linear or perpendicular to the direction of travel.

Initially, a novice can be given a very simple system to look at, perhaps a single Massob, and be encouraged to investigate the effects of the syntonic boost commands. Later, as the learners become familiar with Massobs, they can take more control over them. Using the mouse, children can create new Massobs and give them names. They can vary their size and mass, and give them a velocity. Then they can drag them around the screen, watch them

bump into each other, and make them go into orbits. Finally, they can write programs which affect the behaviour of their creations. Two essential qualities of transitional objects are that they may readily be identified with and that they can continue to be useful, even as the learner's knowledge becomes more sophisticated. In the next chapter we will describe some of our studies of children using Gravitas, and they strongly suggest that learners can indeed make this kind of progression.

2.3 *The Space*

There are a number of features of Gravitas which provide the environment for the Massobs to exist in. We call this environment the space and, rather like a Massob, it has a number of attributes and also a role in the special perturbations method. Strictly speaking, the graphical interface also belongs to the space but we will discuss it separately.

Three obvious attributes of the space are its size, the time step, and the total elapsed time, and they are all on continuous display at the bottom of the screen. The size is actually two things: a size on the screen and a "real" size the space is supposed to represent. In computer graphics these are often called the screen and problem coordinate systems respectively (Foley and Van Dam, 1982). For Gravitas the screen coordinate system is square and only a few hundred pixels on each side. The problem coordinate system though, is as large as the user requires for the task in hand: as big as a laboratory, or the earth, or the solar system, and so on. Gravitas maintains a *mapping* between these two coordinate systems which defines where a Massob appears and how large it seems on the screen.

The time step is of fundamental importance to the dynamic behaviour of Massobs. The product of the time step and a Massob's velocity yields a distance, which Gravitas scales to the space. This becomes the distance the Massob moves across the screen at each step of the animation process. Successive time steps are accumulated into the total elapsed time which Gravitas maintains. The time step also governs the evolution of a Massob's trajectory: at any instant a Massob will be experiencing an acceleration due to the gravitational field of all the other Massobs. This is multiplied by the time step to give a velocity increment at each animation step.

As we mentioned in section 2.2.4, each Massob knows how to calculate the acceleration it will experience from the gravitation of other Massobs in a system. In fact, this capability is only half of the method of special perturbations, as it is realised in Gravitas. The other half requires that each Massob does the calculation exactly once per animation step. This task of synchronisation is carried out by a special method which belongs to the space. Of course, the synchronisation procedure has to run *continuously*, in tandem with the animation process, to maintain the smooth motion of the Massobs across the screen. But one of the basic design goals for Gravitas was that it

should be an unrestricted objectworld, in the sense defined in section 1.6 of chapter 1. This means the user must have uninterrupted access to the programming language, even while the behaviour generating mechanisms are working. Consequently, the animation and synchronisation procedures are run as a *background* task. In other words, the computer keeps them going independently of normal processing.

Just like Massobs, we have given the space a set of commands which operate on its attributes and control its behaviour. Thus we have **time.step** and **set.time.step**, which respectively access and alter the value of the time step. The command pair **elapsed.time** and **set.elapsed.time** do the same thing for the elapsed time. The commands **space.size** and **set.space.size** allow the user to examine and change the size, in metres, of the space. The pair **zoom.in** and **zoom.out** also allow the space size to be altered, but without the user having to think about actual dimensions. Finally, the animation system is controlled by the commands **start.animation** and **stop.animation**, and may be single stepped with the command **step.animation**.

All of these commands, together with those for Massobs, will be listed and described in a little more detail in the next section. As a whole, they form the *programming interface* to Gravitas.

2.4 *The Programming Interface to Gravitas*

We have already mentioned, in the previous two sections, many of the commands which make up the programming interface to Gravitas. Here we will give a complete list, together with any inputs the commands require, and say something about the way in which they may be used in programs. This section is not a comprehensive manual for the system, but it is intended to highlight the similarities and differences between programming in Gravitas and in Turtle Geometry.

2.4.1 *The Command Sets*

For the most part, the command set for Massobs or, indeed, any other objectworld object, derives logically from its attributes. The definition presented in chapter 1 requires "a set of commands that allow the inspection and manipulation of the object's attributes". This stipulation ensures that it is possible to write a procedure for any conceivable operation on a Massob, rather like having a set of spanners and gauges for every job on a car engine. In the case of Turtle Geometry this has led to command pairs like **setxpos** and **xpos**, which respectively set and return the value of the Turtle's x position. As we will see below, we have also constructed a command pair for each Massob attribute. Thus we have **set.mass** and **mass**, and so on.

The rest of the Massob command set comes from the definition's demand that at least "some of these commands should act on the objects in ways that we would expect learners to grasp with little difficulty". These are the four coordinate independent, or body syntonic **boost** commands which we described in section 2.2.3.

The complete programming interface is formed by the addition of the command set for the *space* which the Massobs inhabit. This is logically derived from the space's attributes, which are its size, the constant of gravitation, the time step (which regulates the animation speed), and the elapsed time since the system was started, plus the animation controls **start**, **stop**, **step** and **reset**.

The commands of the programming interface are used in the Logo interpreter which runs alongside Gravitas. This version of the language, Object Logo (Paradigm Software, 1990), allows users to define and execute procedures

in one window, and observe the results in another. Figure 2.8 shows some Turtle geometry being done in this way.

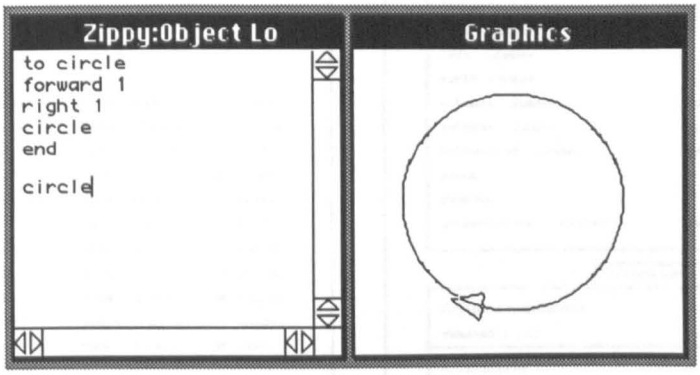


Figure 2.8 Separate windows for procedure definition and Turtle drawing

A conventional programming environment for Turtle Geometry is somewhat simpler. The computer's screen switches between two modes: one allows the construction of procedures, while the other shows the Turtle and its actions.

2.4.2 *Gravitas and Turtle Geometry Programming Interfaces*

Figure 2.9 below shows the complete programming interface to both Gravitas and, for comparison, a typical implementation of Turtle Geometry. The greater size of the Gravitas interface reflects the fact that Massobs (and the space they move within) have more attributes than Turtles (and the screen they exist in).

The important thing to notice about figure 2.9 is the similar structure of the interfaces. At the top level they both divide into a group of commands which deal with the central objects of the systems - Massobs or Turtles, and another group which controls the environment in which the objects are situated - space or screen. Beneath this categorisation we can make another distinction between commands which *affect* some attribute of an object and commands which *return* information about an attribute. As we hinted at the beginning of section 2.4.1, these paired commands allow for very neat programming solutions to a host of problems, some examples of which will be found in Appendix B. After giving some explanatory notes about the commands of the programming interface we will finish off this section with one or two illustrative examples.

MassOb commands

Adjuster commands command *input*	Accessor commands command *input* ⇒ *returned value*
boost *massob*	-
boost.back *massob*	-
boost.left *massob*	-
boost.right *massob*	-
set.boost.strength *number*	**boost.strength** ⇒ *number*
set.standard.boost *number*	**standard.boost** ⇒ *number*
set.pos *massob list*	**pos** *massob* ⇒ *list*
set.xpos *massob number*	**xpos** *massob* ⇒ *number*
set.ypos *massob number*	**ypos** *massob* ⇒ *number*
set.vel *massob list*	**vel** *massob* ⇒ *list*
set.xvel *massob number*	**xvel** *massob* ⇒ *number*
set.yvel *massob number*	**yvel** *massob* ⇒ *number*
set.mass *massob number*	**mass** *massob* ⇒ *number*
set.radius *massob number*	**radius** *massob* ⇒ *number*
set.name *massob name*	**name** *massob* ⇒ *name*
create.massob *name n1...n6*	-
new.massob *name*	-
remove.massob *massob*	-
select.massob *massob*	**selected.massob** ⇒ *massob*

Space commands

Adjuster commands command *input*	Accessor commands command ⇒ *returned value*
start.animation	-
stop.animation	-
step.animation	-
reset	-
set.time.step *number*	**time.step** ⇒ *number*
set.auto.time.step *number when*	**auto.time.step** ⇒ *list*
set.elapsed.time *number*	**elapsed.time** ⇒ *number*
set.space.size *number*	**space.size** ⇒ *number*
set.big.g *number*	**big.g** ⇒ *number*
zoom.in	-
zoom.out	-
-	**cursor.pos** ⇒ *list*
-	**massobs** ⇒ *list*

Turtle commands

Adjuster commands command *input*	Accessor commands command *input* ⇒ *returned value*
forward *number*	-
back *number*	-
left *number*	-
right *number*	-
setxpos *number*	**xpos** ⇒ *number*
setypos *number*	**ypos** ⇒ *number*
setheading *number*	**heading** ⇒ *list*
penup	-
pendown	-
setpencolour *number*	**pencolour** ⇒ *number*

Screen commands

Adjuster commands command *input*	Accessor commands command ⇒ *returned value*
clearscreen	-
wrap	-
nowrap	-
setbackgroundcolour *number*	**backgroundcolour** ⇒ *number*

Note: Real Turtle Geometry implementations normally contain more commands than we have listed above. For instance the turtle may be able to set its pen pattern or be asked to move towards a particular coordinate. The commands we have given are simply a representative core set.

Figure 2.9 Programming interfaces for Gravitas and a typical implementation of Turtle Geometry

2.4.3 *Notes on the Programming Interface Commands*

Before moving on, there are a few things we should say about the commands which make up the programming interface shown in figure 2.9. The boost commands have already been covered but the boost *strength* needs a comment. The user can set this to any value in units of metres per second (remembering to think of the boost as a velocity increment rather than an acceleration) using the **`set.boost.strength`** command. Alternatively, Gravitas can be asked to calculate a "reasonable" boost, where reasonable means a value that will produce a visible effect on Massobs, given the current size of the space and the time step. The command which does this is called **`set.standard.boost`**, and it is used to set a default value when Gravitas is first loaded, so that the whole issue of boost strengths may be ignored when learners first meet the system.

The next feature of note is that the position and velocity may be set either by separate x and y commands, or by the single commands, **`set.pos`** and

set.vel, which take a Logo *list* as their second input. The corresponding accessor procedures, **pos** and **vel**, return a list when called. These functions are simply an extra convenience and they parallel similar commands which are part of most Turtle Geometry implementations.

The pair of commands **set.name** and **name** can be used to alter (and inspect) the name a Massob displays on the screen. **set.name** should be used with circumspection because it can break the correspondence between a Massob's variable name and its displayed name, making programs less clear.

new.massob is the primitive command which underlies **create.massob**. It simply creates a new Massob without setting any of the attributes. Normally, users would not find any need for it. However, **remove.massob** which, as its name suggests, removes a Massob may well be useful. The final Massob command **select.massob**, determines which object has its attributes printed in the displays of the graphical interface.

The space commands, **start.animation**, **stop.animation**, and **step.animation**, are mostly unproblematic, but **reset** deserves explanation. All Massobs remember the values of the attributes they were created with. When asked to **reset** they restore these values and redisplay themselves. The next command, **set.time.step**, is straightforward, but a glance at **set.auto.time.step** shows it to have two inputs - *number* and *when*. Gravitas can be asked to control the time step to ensure that Massobs do not immediately fly off the screen. **set.auto.time.step** examines every Massob to find out which is moving fastest. It then calculates a time step which will cause that Massob to move *number* pixels at each step of the animation process. If the second input, *when*, is set to **"now**, the process is carried out just once. If it is set to **"always**, the calculation is performed every step, which can be useful when a system contains Massobs which are experiencing large accelerations. However much the time step is altered, Gravitas continues to keep an accurate record of the total time a system has been running and this may be set or examined with the **set.elapsed.time** and **elapsed.time** commands.

By default, Gravitas uses the accepted value of 6.673×10^{-11} Nm^2kg^{-2} for the Universal Constant of Gravitation, and it is stored in the command **big.g**. However, this may be changed with **set.big.g** if users wish to experiment with other values. The size of the space may be controlled by two more

commands **set.space.size** and **space.size**. For novices to the system though, **zoom.in** and **zoom.out** which respectively halve and double the dimensions of the space, are easier to use. Finally, the command **massobs** returns a list of all the currently defined Massobs, which Gravitas keeps up to date as objects are created and removed.

2.4.4 *Example uses of the Programming Interface*

As an example, imagine we want to build a procedure which doubles the velocity of a Massob. The following code performs the operation:

```
to  double.velocity  :massob
   set.xvel  :massob  2  *  xvel  :massob
   set.yvel  :massob  2  *  yvel  :massob
end
```

The procedure **double.velocity** takes a single Massob as input. Its first action is to use the *adjuster* command **set.xvel** to set the x component of the Massob's velocity to twice its existing value. The process is then carried out for the y component.

For a second example we will consider the phenomenon of *escape velocity*. If a projectile leaves the surface of a planet with sufficient speed it will never return, no matter how long we wait. The lowest speed at which this occurs is called the escape velocity and it is given by the formula: $v_e = \sqrt{\frac{2Gm}{r}}$, where G is the gravitational constant, m is the mass of the planet, and r is its radius. We can translate this into a Logo procedure using Massob and space *accessor* commands:

```
to  escape.velocity  :massob
   output  sqrt  (2  *  big.g  *  (mass  :massob)  /  radius  :massob)
end
```

A couple of tries with Massobs we have already seen (in Chapter 1):

```
print  escape.velocity  :io
   1807.49

print  escape.velocity  :jupiter
   59580.7
```

So, the escape velocity for Io is less than 2 kilometres per second, or about three times the speed of Concorde. Jupiter, much more massive, has an escape velocity of almost 60 kps. For comparison, the Earth's escape velocity is about 11 kps.

2.5 *The Graphical Interface*

We have described the programming interface first in order to highlight the structural resemblances between Gravitas and Turtle Geometry. However, newcomers to the system are normally introduced to the graphical interface first, as it is far easier to learn.

The graphical interface to Gravitas duplicates the *functionality* of the programming interface, in that it provides controls and numeric displays for each of the items listed in figure 2.9. These can be seen in figure 2.10. The work of the adjuster commands is done by buttons which allow each attribute of the space, and the Massobs within it, to be set. The accessor commands are replaced by small 'data windows' which show the values of the attributes.

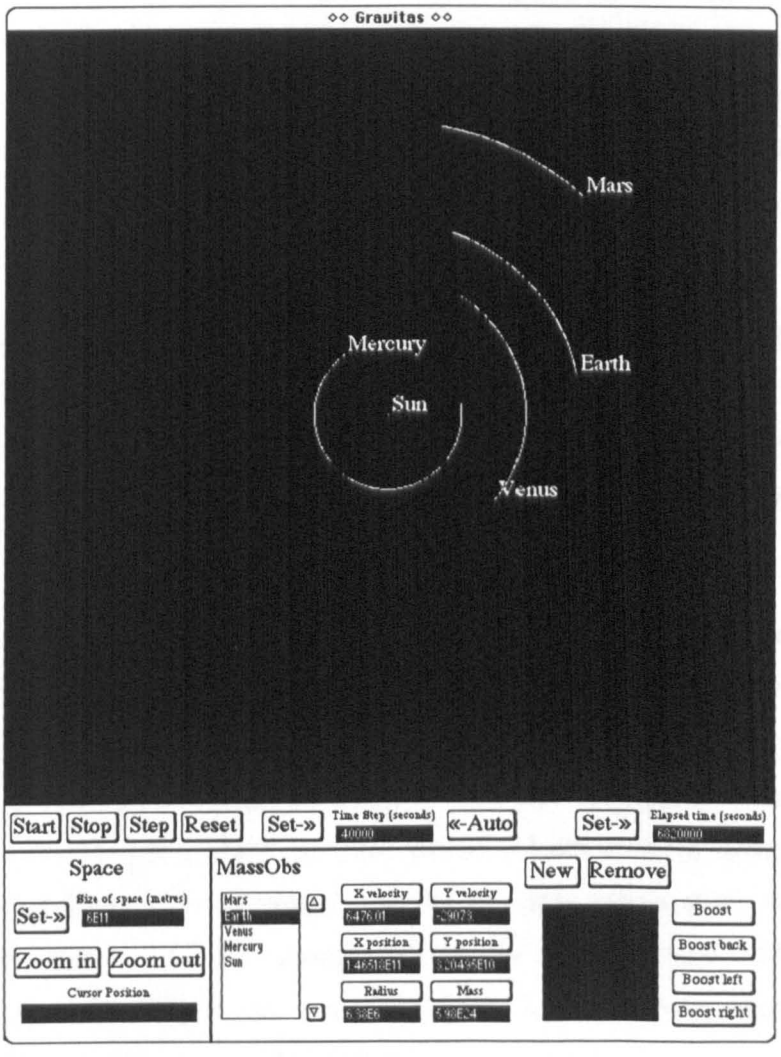


Figure 2.10 The Terran Planets

The strip which runs immediately underneath the space contains buttons which control aspects of the animation: the first three start, stop and step the animation process. The Reset button clears the screen and sends all the

Massobs back to their original positions. Next, a data window showing the current value of the time step is flanked by two buttons, one for setting the step directly and another for invoking the automatic mechanism described in the last section. Finally, a button and a data window allow the total elapsed time to be set and read.

At the bottom left of figure 2.10 are the buttons which control the size of the space. The current value is shown beside the Set button. Another data window displays the position of the cursor in space coordinates when the mouse button is held down. This feature allows the user to measure things on the screen, such as the diameter of an orbit or the position of an impact.

To the right of the space buttons is the Massob controller. The currently defined Massobs are all listed in the scrolling window on the left which is the analogue of the accessor command **massobs**. Clicking on a Massob in the list selects it, and in figure 2.10 the Earth is the **selected.object**. Consequently the values of the Earth's attributes are displayed in the six numeric displays immediately right. Each numeric display has a corresponding button which allows the relevant attribute to be set.

The New button allows new Massobs to be created, which then appear in the square data window underneath until all of their attributes have been set and they move into the space. The Remove button erases the currently selected Massob. Finally, at the far right lie the cluster of four Boost buttons, which also act on whichever Massob is selected.

In the present implementation of Gravitas, some of the functions of the programming interface do not appear in the main window. Instead the button equivalents of **set.boost.strength**, **set.standard.boost**, **set.name**, and **set.big.g**, and their corresponding accessor commands appear in a concealed auxiliary control panel which can be brought into view by a menu operation. The original reason for this separation was to protect novices from some of the complexities of the system. However, our experiences of using Gravitas with children indicate that this is probably unnecessary and so this is a feature which may change.

2.6 *The Direct Manipulation Interface*

We now describe a third and final way of operating Gravitas. This last interface is not a complete substitute for either of the others, but it does render the construction of Massobs even easier than pressing buttons. It allows users to set the six physical attributes of a Massob *directly*, without recourse to buttons and with no need even to type in numbers.

Shneiderman, (1982) introduced the term Direct Manipulation to describe interfaces which allow the user to operate on a convincing representation of the material of interest, using straightforward actions rather than indirect commands. The obvious examples are 'What You See Is What You Get' editors which present an image of a document on the screen that is as close as possible to the printed appearance, and which allow editing to be carried out with the mouse. In this sense, Gravitas has a Direct Manipulation interface to Massob attributes.

First of all, the position of a Massob can be set by "picking it up" with the mouse and dragging it around the screen. Next, while a Massob is in the square data window at the bottom right of the control panel, its velocity may be set by "rubber banding" a vector with the mouse. That is, while the mouse button is held down a line is drawn from the centre of the Massob to the cursor position and the length and direction of this line are used to compute a velocity. Lastly, if the option key is held down at the same time as the mouse button, then instead of setting the velocity it is the Massob's radius which is varied. The Massob grows and shrinks as the mouse is moved away from and nearer to its centre. Doing this also varies the Massob's mass, using its current density to calculate the value.

The Direct Manipulation interface does not *replace* the buttons and data windows, it augments them. It is acknowledged (Hutchins et al, 1986) that Direct Manipulation interfaces have their limitations, the most serious of which, for Gravitas, is the problem of representing variable quantities with precision. The attributes of Massobs can range over many orders of magnitude, from a tiny particle to a giant star. For example we may wish to define a satellite weighing a few tens of kilograms and a planet whose mass is around 10^{25} kilograms, and both of these quantities may be known with great precision. This kind of situation is unwieldy for the Direct Manipulation approach, and whenever accuracy is required in Gravitas, users will employ the buttons of

the graphical interface or the programming commands. Figure 2.11 shows the Massob commands which do have Direct Manipulation equivalents.

MassOb commands

Adjuster commands	Direct Manipulation Equivalent
`boost`	-
`boost.back`	-
`boost.left`	-
`boost.right`	-
`set.boost.strength`	-
`set.standard.boost`	-
`set.pos` **`set.xpos`** **`set.ypos`**	MassObs may be dragged into position with the mouse.
`set.vel` **`set.xvel`** **`set.yvel`**	While a massOb is in the small window, a velocity vector for it may be drawn with the mouse.
`set.mass` **`set.radius`**	In the small window, the radius may be set with the mouse. This indirectly sets the mass because density is held constant.
`set.name`	-
`create.massob`	-
`new.massob`	-
`remove.massob`	-
`select.massob`	Clicking on a massOb selects it.

Figure 2.11 Direct Manipulation Equivalents for Massob adjuster commands.

2.7 *The Utility of Multiple Interfaces*

Why, it is reasonable to ask, have we gone to the trouble of installing two functionally identical ways of managing the system? The proper answer to this question will be given in chapters 3 and 4, but we will summarise it here.

Originally, there was no graphical interface to Gravitas. The system was operated entirely through programming interface commands typed into the Logo interpreter just like conventional Turtle Geometry. It was understood, of course, that this mode of operation placed a hurdle in the way of novices, and the relatively large number of commands together with the need to consider the inputs to those commands was seen to have a deterrent effect on users of the prototype. In fact a similar, though less severe, phenomenon has been observed for Turtle Geometry, and in consequence various attempts have been made to "lower the threshold". For instance the researchers on the Craigmillar Logo Project (Hughes and MacLeod, 1986) provided young children with a system called STARTER. This allowed them to drive a Turtle with single presses of the keys F, for `forward` a set amount, B for `back`, L for `left 90` and R for `right`. After a few familiarisation sessions the researchers added more features to STARTER until the children were, in effect, programming the Turtle. They stopped short, however, of actually getting the children to define true procedures.

Inspired by examples such as the above, we decided to create an easy to use, "instant" interface for Gravitas, and the present graphical interface is the result. We envisaged that this would be the medium for a user's initial work with the system but that it would give way to the construction of programs as confidence increased.

However, we have discovered that while the amount of programming does rise with familiarity, there is little or no reduction in the use of the graphical interface. In fact there seems to be a synergistic effect between the two interfaces which allow students to take on more complex projects than would otherwise be the case.

For example, we have studied children developing Logo programs that control the system (via the programming interface) to achieve a long term goal, such as getting a rocket from the Earth to the moon and back. There is no practical *analytic* formula for this journey, but, with appropriately timed

boosts, Gravitas can carry it out. In principle, the mission could be developed using either one of the interfaces on its own, but in practice it would be very difficult. The reason for this is that the journey has to be split into stages: a launch phase, then a boost into a circular orbit, a transfer orbit to the moon, and so on. The boosts which control these stages have to be accurately controlled both in their strength and their timing. Used together, the graphical interface allows the rapid testing of varied boost strength and timing at the various critical points in the mission, while the concreteness of programming allows correct sections of the mission to be "frozen" into procedures so that the complete solution may be approached incrementally.

2.8 *Alternative Syntonic Commands*

As we hinted in section 2.2.2, the boost commands **boost**, **boost.back**, **boost.right** and **boost.left** are not the only way syntonic commands could have been implemented for Massobs. They do permit a simple explanation: boosts always act either along the line of, or at right angles to a Massob's trajectory. This means the learner does not have to think so hard about the direction they are applying accelerations. A further reason they were selected was because they proved very useful in orbital mechanics.

However, in other settings, for instance a system where gravitation may be considered insignificant, other syntonic commands might be thought appropriate. One such set of commands installed in Gravitas added another attribute to Massobs - a *boost heading* indicated by a small arrow pointing from the Massob's centre - which is independent of the direction in which the Massob is travelling. Naturally, this new attribute had commands to set and inspect it: **set.boost.heading** and **boost.heading**. But the relative commands **rotate.boost.left** and **rotate.boost.right** were also added. With this arrangement, Massobs behave like the Dynaturtles described by diSessa and White (1982) although Gravitas allows them to be programmed while still moving and, of course, if there are two or more Massobs, and they are massive enough, then gravitational forces will visibly affect their motion. Figure 2.12 illustrates the simplest case.

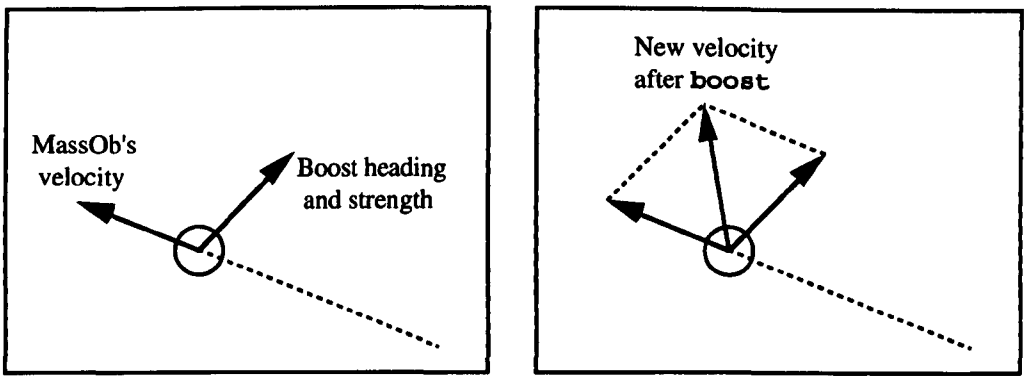


Figure 2.12 The alternate boost commands used in a prototype of Gravitas

Another version of the boost commands which we tried out differed slightly in the implementation of **boost.left** and **boost.right**. Figure 2.13 below (based on a magnified detail of figure 2.6) shows that the standard perpendicular boosts actually increase the *speed* of a Massob as they rotate its velocity vector. In other words, the velocity vector gets longer:

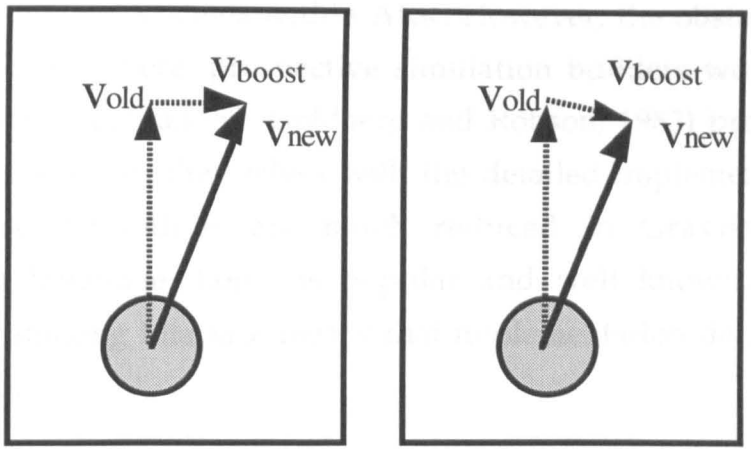


Figure 2.13 The standard **boost.right** and a speed conserving version used in a Gravitas prototype

The right hand diagram shows a modified **boost.right** which actually leaves the speed unchanged. Due to lack of time, we did not try this form of the commands in any of our formal studies. However, a comparison of the different schemes might be an interesting experiment for the future.

A key feature of Gravitas is that anyone who is reasonably proficient in Logo can implement other kinds of boost, or entirely different syntonic commands, because they are built on top of the general commands for manipulating the attributes of Massobs. In this sense, Gravitas is an *open system*. For instance, imagine we want to arrange that a Massob uses up "fuel" with each boost, like a real space vehicle. This could be accomplished:

```
make  "boost.mass  100
make  "minimum  1000

to  new.boost  :vehicle
    if  mass  :vehicle  <  :minimum  [print  [Out  of  fuel!]
                                      stop]
   boost  :vehicle
   set.mass  :vehicle  (mass  :vehicle)  -  :boost.mass
end
```

Figure 2.14 Logo code to implement a fuel using boost.

The constants **boost.mass** and **minimum** state how much fuel a boost uses and how much the empty vehicle weighs. Once the procedure **new.boost** is defined it can be used just like any other command.

Not all users of Gravitas would be able to extend it in this way because of the level of programming required. In the same context, Smith (1987) identifies two classes of user for his system, the Alternate Reality Kit. - "The *applications-level* user might typically be a student carrying out a simulated lab. At a lower level the *simulation builder* is the creator of a particular application". Clearly, Smith hoped that people would be able to build their

own components and systems within ARK. However, the obstacles in the way of this were quite severe: prospective simulation builders would need to be relatively skilled Smalltalk-80 (Goldberg and Robson, 1983) programmers and would have to concern themselves with the detailed implementation of ARK objects. These difficulties are much reduced in Gravitas, where the programming language, Logo, is popular and well known, and the well defined programming interface means that implementation details of Massobs can be ignored.

2.9 *Summary*

We have described Gravitas, a Discovery Learning Environment for the topic of Newtonian Gravitation which allows students to examine the dynamics of gravitating objects called *Massobs*, which move within a two dimensional *space*.

Massobs are completely defined by the values of their *attributes*: position, velocity, size, mass and name. Users are free to alter these attributes directly but another set of commands, which apply boosts to Massobs and thereby change their state *indirectly*, have been installed in Gravitas. These *syntonic* commands are intended to provide links to sensori-motor knowledge about pushing which even young learners will bring to the system. They parallel the well known Turtle Geometry commands for which Papert has provided a similar justification.

Massobs also possess mechanisms which generate their particular behaviour. Specifically, they have a method by which they can compute the effect on their trajectory of the gravitational pull of all other Massobs. The Method of Special Perturbations, as it is known, is a numerical technique which allows us to circumvent the lack of general analytic equations of motion for systems composed of three or more masses.

Papert has often described the Turtle as a *transitional* object which can help children cross the gulf between their personal everyday knowledge of the world and the formal systems we expect them to learn. We have followed his example and characterised Massobs as transitional objects for some of the concepts of Newtonian physics. In this sense, Massobs are partners for the Turtle.

Gravitas has been equipped with two functionally equivalent interfaces. The first is a straightforward programming interface consisting of Logo extensions which are analogous to the commands of Turtle Geometry. The second interface is purely graphical, operated by mouse clicks and drags. It combines the Direct Manipulation of Massobs, for ease of use and speed, with buttons and numeric displays for precision. Although the graphical interface was originally added to Gravitas purely to increase its ease of use, during practical studies we have found indications that a synergy between the two

interfaces allows learners to make better progress than would be the case were only one present. We will have more to say on this in chapter 3.

Finally, we have shown that Gravitas is an *open* system, which anyone proficient in Logo programming may augment or modify. For example, new syntonic commands, which replace the standard boosts, may be installed with little difficulty.

3 Gravitas in use

3.1 Overview

In this chapter we describe observations we have made of learners using Gravitas. To date, we have carried out studies of six school students aged between 13 and 18. The subjects were videotaped as they worked at the computer assisted by the researcher. In two of the studies the students worked alone, while the other four students worked in pairs. Each study typically lasted five to six hours.

We have also made informal studies of seven adults. These subjects each used Gravitas for about two hours. Notes were taken during their sessions and the system transcript of their actions was preserved, but they were not videotaped.

These studies were not intended to measure cognitive changes in learners, or to contrast the "Gravitas way" of learning a physical concept with a traditional approach. While many such experiments have been carried out for Turtle Geometry, (for instance (Clements and Gullo, 1984) and (diSessa and White, 1982)) it was felt to be premature for such a programme to be tried with Gravitas. We wished to begin by examining the nature of the educational activities Massobs could support. We knew that Massobs could be manipulated by hand or by Logo programs, and that they could be assembled into gravitating systems. These were the aims of the design. But we wanted to find out what kinds of systems learners actually *could* build, and what sort of programs they could write.

In many ways this aim parallels some of Papert's earliest investigations with the Turtle. For instance in *Twenty Things to do with a Computer* (Papert and Solomon, 1971), the authors describe a range of possible uses for the Turtle, many of which have been the departure points for subsequent research. Although the paper contains anecdotal information about children engaged in these exercises, there is no discussion of cognitive effects and no formal comparison with traditional methods of teaching. Rather, Papert and Solomon set out to show the scope of the new computational object.

Our approach differs from Papert and Solomon's in that we will be describing relatively few activities, but in greater depth. In fact, we have chosen

to transcribe, with detailed annotations, video recordings of just three different activities: The construction of Massob systems, the explanation of their physics, and the writing of programs to manipulate them.

However, at an early stage of the studies we began to notice two phenomena which shaped the investigations thereafter: first of all, we found learners were often surprised by the long term behaviour of Massob systems *of their own construction*. Second, we observed that users were taking advantage of a synergy between the graphical and programmable interfaces, which enhanced the human-computer interaction and allowed the users to take on more complex tasks than would otherwise have been the case. The annotated transcriptions which follow will highlight these phenomena.

3.2 *Constructing a System*

3.2.1 *The Task*

Subjects were videotaped as they used Gravitas to work on tasks set by the researcher. As an introduction to the system the subjects were shown a pre-defined orbital system (see figure 3.1a) and they were given between 5 and 10 minutes to familiarise themselves with the various buttons and displays. They were then presented with an empty space and asked to construct a system representing the Earth being orbited by the Moon. Figure 3.1b shows the appearance of the empty Gravitas space they were given.

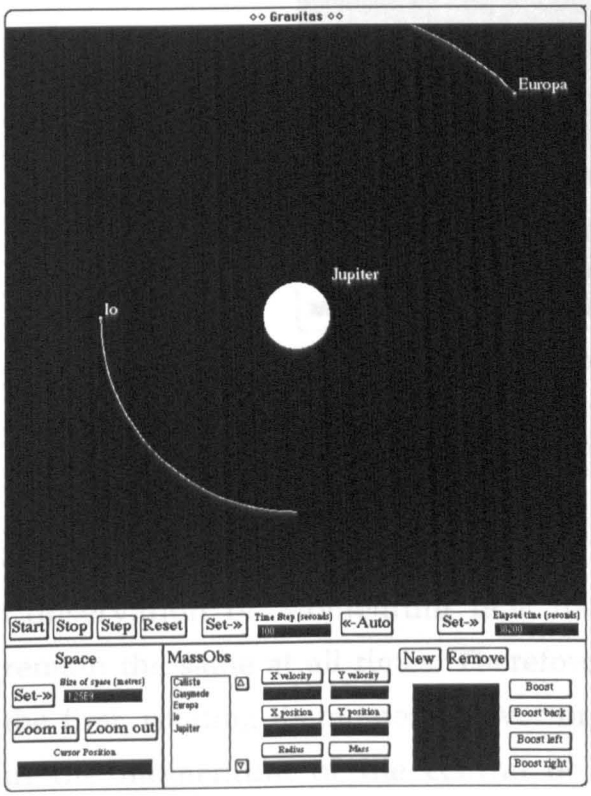


Figure 3.1a A typical orbital system

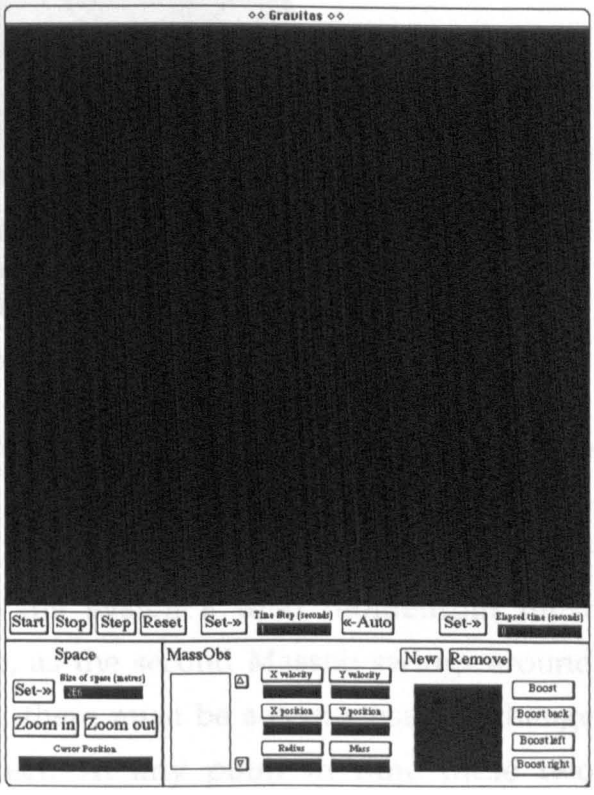


Figure 3.1b An empty space

From previous experience we expected the subjects to be surprised by the behaviour of the system they constructed. The reason for this lies in the method people commonly choose to build orbital systems. They put one *stationary* Massob at the centre of the space and then position the second Massob some distance away, with a velocity tangential to the first. When, either by trial and error or by calculation, they get this tangential velocity correct, the result is a circular orbit. Or so it seems at first. Sooner or later, depending on the mass ratio of the two objects, this orbital system begins to move through space, in the same direction as the second Massob's initial velocity. The central Massob, which was created to be stationary, has taken on a

velocity. Figure 3.2 shows an example of this orbital procession, which usually surprises even those users with a solid background in school physics.

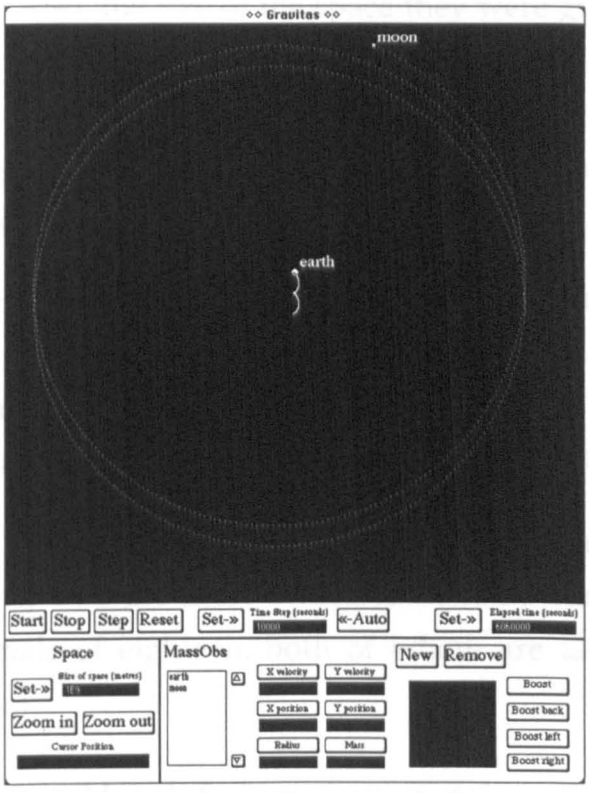


Figure 3.2 Orbital Procession

The procession occurs because of the initial conditions the subjects give to the two bodies they create. The initial momentum vector of the *system* is due entirely to the second Massob. In the absence of external forces the Law of Conservation of Momentum tells us the system's total momentum must remain the same at all times. Therefore, as the second Massob swings around the first, rotating its momentum vector, there must be a compensatory change in the momentum of the central object. At any point in time these two momentum vectors must add up to the original quantity.

The resolution of this surprise formed the second part of the task set to the subjects. We were not looking for a clear, formal explanation along the lines we have given above, because some of our subjects were not very familiar with the Law of Conservation of Momentum. Furthermore, those subjects who had learned about the Conservation Law had done so in rather abstract, rectilinear contexts where there are no forces acting. Instead, we wished to see if they could construct an initial state such that procession no longer occurred.

To help them start the task, subjects were given a brief table of relevant astronomical data - the mass and radius of the Earth and the Moon, and their average separation. Also, the size of the space they were given was deliberately set to that of the Moon's orbit. All of the subjects, including those with very little background in maths, were familiar with the idea of x and y coordinates for position, although as we will see some confusions arose when they tried to consider the x and y components of a velocity.

The researcher's role in the studies was not passive. Guidance was given at several points with the intention of encouraging the subjects to make the major insights for themselves. In the transcripts which follow, all the important remarks and hints offered by the researcher are included.

We begin with a transcript of a pair of users carrying out the task. The pair chosen are those who had the most difficulty in completing it, a sixteen year old male and a female of eighteen, both of whom are taking predominantly arts subjects at school.

3.2.2 *Creating a Massob System: Transcript 1*

After a ten minute introduction to Gravitas the subjects were asked to begin the task. They quickly found the **New** button and used it to create a Massob which they called "Earth". This was dragged out of the factory into the centre of the space. The **New** button was used again to create another Massob called "Moon" and this was also dragged onto the screen, up and across from the Earth.

Tim: "Do you have to do it like real life?"

Researcher: "Yes, they have to be the real numbers."

Tim: "So we have to work out the radius on these? [points at the buttons] We click on the radius and type it in?"

At this point they decide that one of them, Tim, will control the mouse and that Nicola will operate the keyboard. Tim clicks on the **Radius** button, Nicola reads the value for the Moon from the table and types it in. They repeat the process for the Moon's mass:

Nicola: "So we do the same for the Earth now?"

R: "Yes, just the same."

They set the radius and mass of the Earth then Tim clicks on the **Start** button. The Massobs begin to move. Figure 3.3

N: "They're heading straight for each other!"

R: " So why do they do that?"

T: "It's the gravitational force pulling them together."

R: "But the Moon doesn't fall in real life. What keeps the Moon up there?"

T: "Oh dear, what's it called when it goes round... so, do we have to put the force in there?"

R: "Not a force."

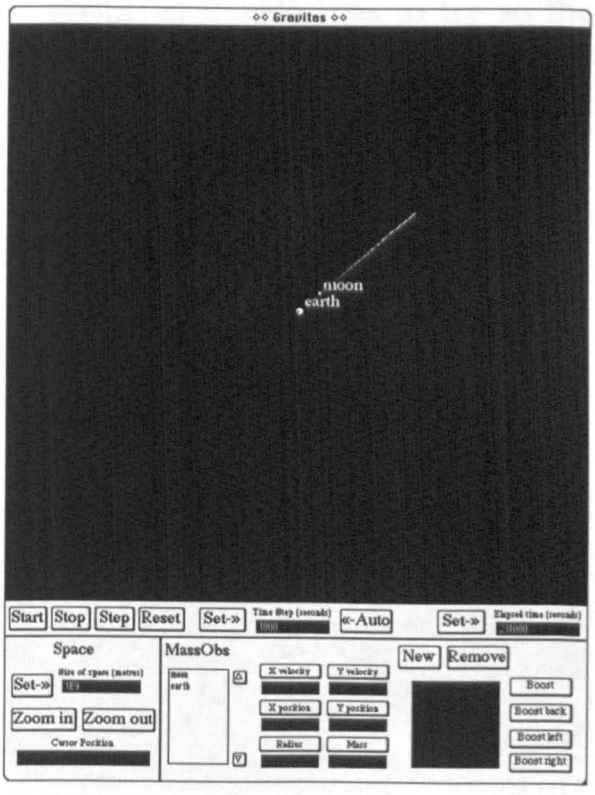


Figure 3.3 Earth-Moon system with no orbital velocity

They decide to put in a velocity of 2000 metres per second in the x direction (i.e. *away* from the Earth) believing this will balance the Earth's attraction. The Moon flies rightwards. Figure 3.4

N: "It's going off the page isn't it? Stop it."

R: "So what do you think about 2000?"

N: "A bit big."

R: " But what about the direction?"

T: "Ah, we haven't set the y one have we? We've got to set both of them. It's got to be the same hasn't it, I guess. No, that'll do a square. If they're both zero that'll keep the same distance, er..."

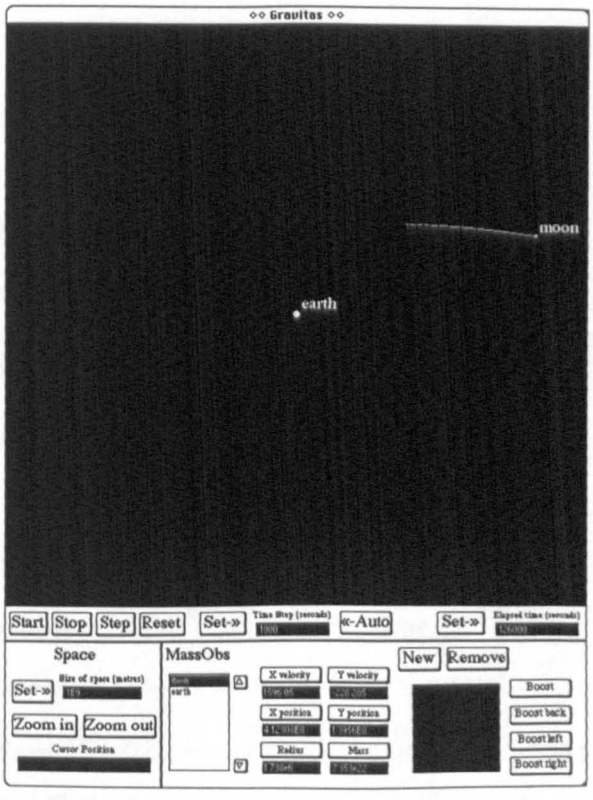


Figure 3.4 Moon with initial x-velocity $2000ms^{-1}$

They appear to be rather confused about what velocities to try so the researcher makes a suggestion:

R: "If you point on the screen where you think you want it to go then it's easier to work out what the velocities should be."

T: [Points out a tangent] "x about 100, go down about 150. No, we want -150."

They start the system. Moon falls toward Earth again after a small wiggle.

R: "What conclusion do you draw from that?"

N: "We need the number going that way bigger. The x number."

They set x vel to 250, y to -150. It falls in.

N: "It's a bit better."

They set x vel to 1000, y to -250. Figure 3.5

N: "That's a good circle, if not big enough. It's better."

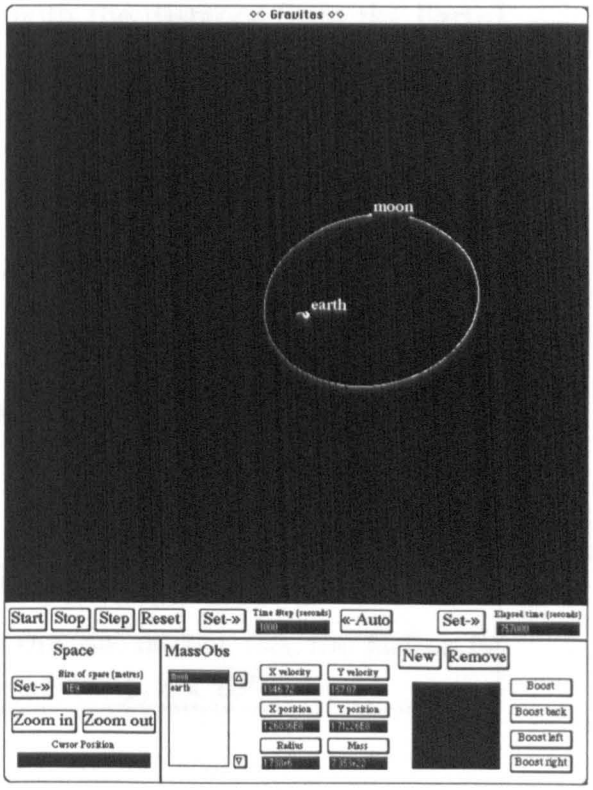


Figure 3.5 Initial x-vel 1000ms^{-1} , y-vel -250ms^{-1}

At this point it becomes apparent that Tim is confusing velocity and position so the researcher points this out to him. Tim quickly realises his mistake. The researcher also gives them another hint:

R: "Can you think of a way to make your job simpler? You have the Moon up and away from the Earth at an angle so you have to set two velocities all the time."

N: "So if we moved the Moon down next to the Earth..."

R: "How does that help you?"

N: "Then we don't have to do the x."

T: "We could also do it there [points to 12 o'clock] and wouldn't have to do the y one."

They use the **Y Position** button to set the Moon's y coordinate to 0, level with the Earth. Both subjects begin to feel that there should be a better way of proceeding than trial and error:

T: "Is there a formula?"

R: "Yes..."

N: "Has it got something to do with the distance from the Earth to the Moon?"

T: "And its mass?"

R: "Yes, but you also have to know how long it takes for the Moon to go round the Earth."

T: "Do we have to work out pi?"

N: "I think we should try working out the formula..."

T: "What's pi?" [he is having trouble attracting the researcher's attention]

N: "The circumference is..."

R: "The formula for the circumference is $2\pi r$."

T: "We're going to have to work out the radius. Oh, the radius is the distance [looks at the table of data] , 4×10^8."

Nicola uses the calculator to work out $2 \pi \times 4 \times 10^8$: the circumference of the Moon's orbit

R: "That's the distance around the Moon's orbit and it takes 28 days."

T: "So it's 28 days divided by that."

R: "That divided by 28, but we want it in seconds."

Nicola uses the calculator to work out $28 \times 24 \times 60 \times 60$, the Moon's orbital period in seconds, then divides the circumference by it to get $1038.8864 ms^{-1}$.

T: "So we'll put that in the y-velocity."

N: "Round it up because it won't all go in there." [the displays look too small]

The Moon moves in an anti-clockwise ellipse around the Earth. Figure 3.6

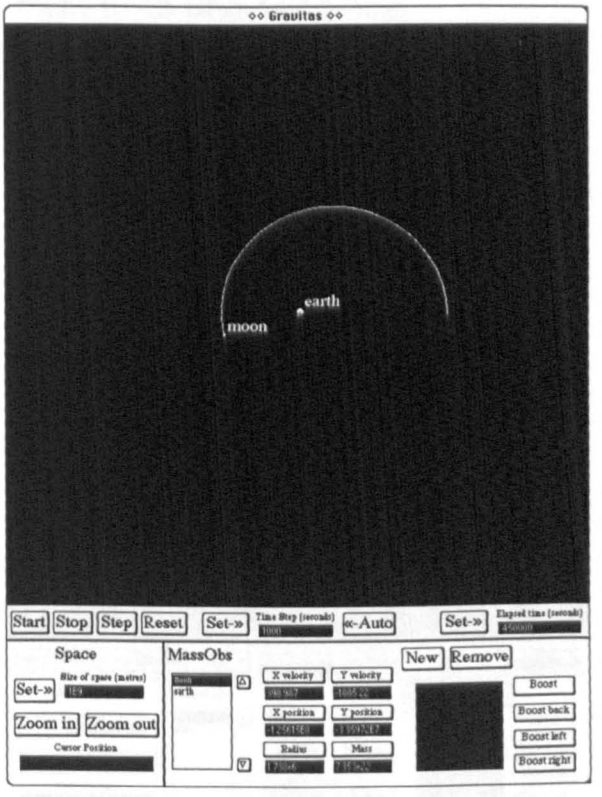


Figure 3.6 Moon's initial position wrong.

R: "Can you imagine why it's not right?"

T: "We haven't set the distance between Earth and Moon!"

They set the distance to the correct value and get a circular orbit. At this point they are surprised to find that the system they have built moves up the screen. Figure 3.7.

T: "The Earth is moving. It shouldn't be moving at all!"

N: "Why is the Moon doing that? Why is the Moon going different circles?"

They continue to express their puzzlement as the Earth and Moon progress up the screen.

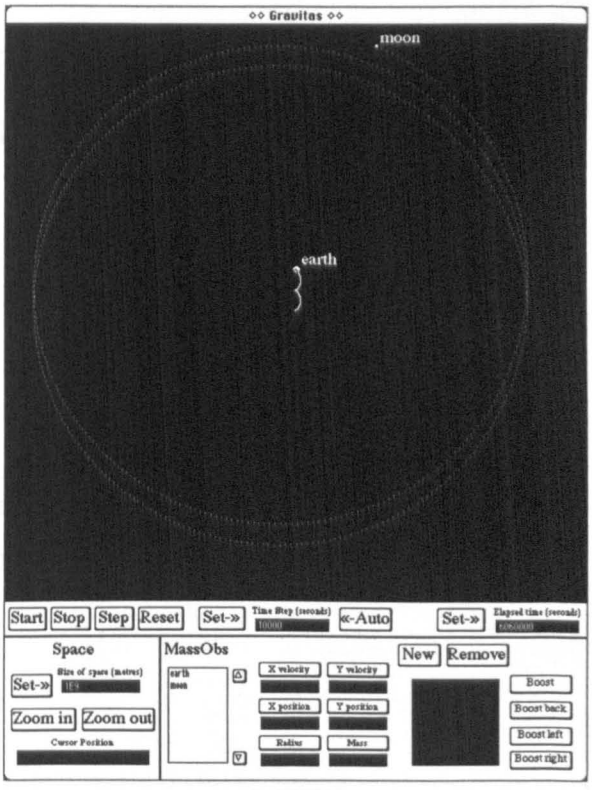


Figure 3.7 The orbital procession 'surprise'.

After allowing them some time to ponder the unexpected phenomenon, the researcher begins an attempt to lead them to an understanding:

R: "Think about what's going on as the Moon goes from the three o'clock position to the twelve o'clock. Imagine you were swinging a bucket around on your arm, what would you feel?"

T: "Pressure... getting heavier at the bottom and lighter at the top."

R: "But there is no bottom or top. This is space. It helps to think of quarter turns."

R: "What is causing the Moon to move like this [indicates its curved orbit]"

T: "Gravity."

N: "So is the Moon's pulling the Earth?"

R: "Yes, that's part of it."

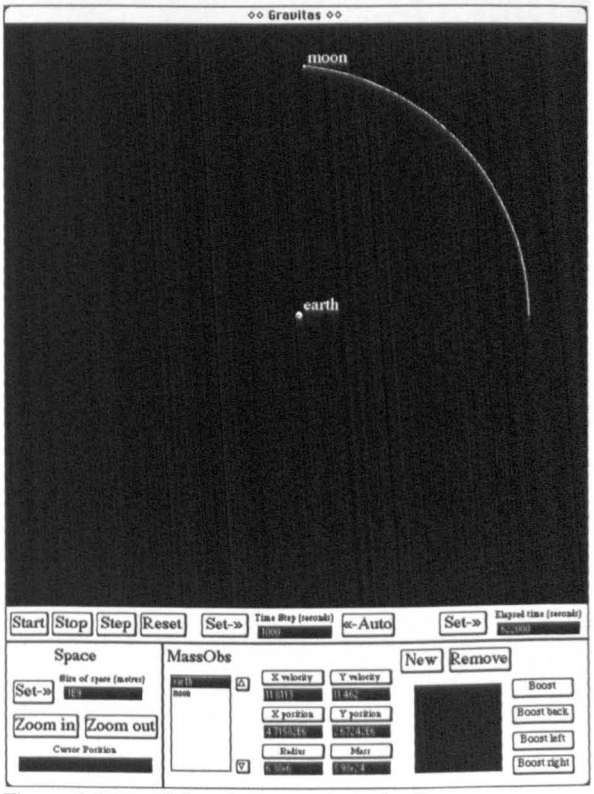


Figure 3.8 Resolving the 'surprise': the first quarter.

T: "So the Earth isn't big enough?"

R: [Laughs] "Well, if the Earth was infinitely big it wouldn't move."

N: "We can't change its mass though."

The dialogue hints at the problem of applying sensori-motor knowledge, which is learned on the Earth's surface, where gravity acts downwards and our feet are solidly placed, to motion in space. The researcher begins to lead them to discover a way of stopping the system from processing up the screen.

R: "Can you think of a way of keeping them in the same place?"

T: "Make them further apart?"

R: "No, that won't work."

They decide they want the Moon to orbit Earth clockwise so they reverse the initial y-velocity.

R: "Go from 3 o'clock to 6. Now where do you think it will be pulling the Earth?"

T: "Downwards." [Figure 3.9]

N: "It's moving in a little circle on its own." [points at the Earth]

R: "It's not actually a circle. Can you think of a way of turning that into a circle?"

N: "Setting its value so it goes round in a circle."

R: "Which particular value?"

N: "The Earth's velocity."

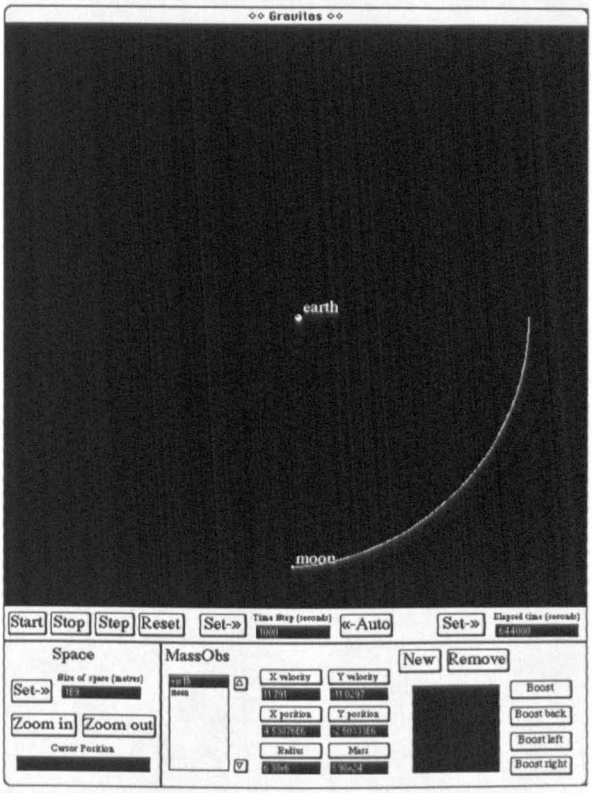


Figure 3.9 Resolving the 'surprise': clockwise orbit.

T: "Earth's velocity. The Earth's velocity is not very strong so the Moon is taking it. It should have a higher velocity."

N: "So its velocity..."

T: "The Earth's velocity should be the same as the Moon."

R: "But look how fast the Moon is going."

T: "Let's try 100."

They give the Earth a y-velocity of $100ms^{-1}$. It travels upwards. Figure 3.10a.

N: "That's totally wrong."

They decide to try $1ms^{-1}$.

N: "No... same as before..."

The motion is almost the same as their original try. Figure 3.10b.

They decide to try $10ms^{-1}$.

N: "Tiny little circle..."

R: "That's pretty much it. [Figure 3.10c]"

$10ms^{-1}$ is quite close to the actual value of 12.7 which will produce equilibrium.

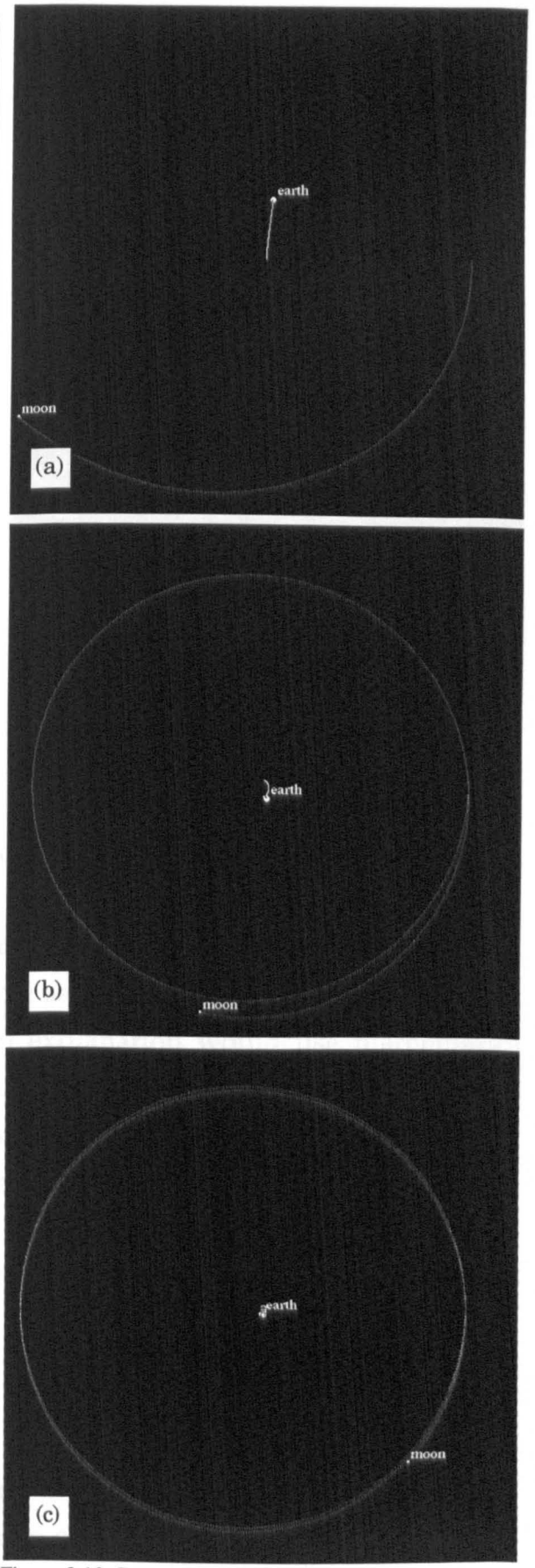


Figure 3.10 Getting Earth's y-velocity right.

The researcher explains to them that it is possible to calculate the required velocity using the formula $v_e = -\left(\frac{v_m m_m}{m_e}\right)$

They use the calculator to do this and create a system in equilibrium. Figure 3.11.

R: "Can you explain in your own words why it is staying in one place?"

T: "It's staying in line. Its like two little circles and when the Moon is on that side the Earth is on the other.

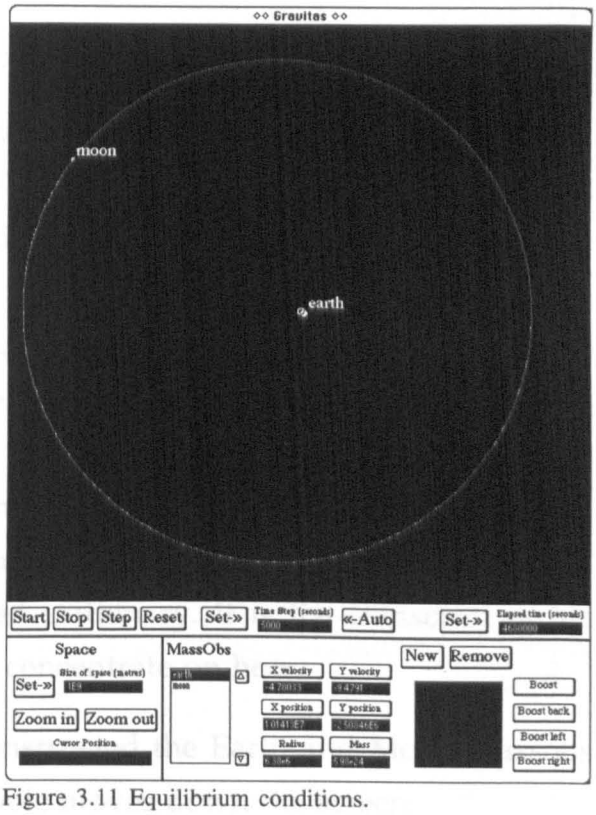


Figure 3.11 Equilibrium conditions.

3.2.3 *Discussion*

The session we have describe above took around one hour and fifteen minutes to complete. In a sense the subjects have "solved" the problem. However, their solution is very different from the kind we are used to in physics. By trial and error (and with some assistance) they found a way to fix a "bug". Of course, a traditional, formal explanation would use mathematics and, like our discussion at the beginning of this section, be expressed in terms of momentum conservation and the system's initial conditions. Nicola and Tim have not made the insight that the total momentum of a system remains constant over time, and therefore only an initial momentum of zero can give rise to a stationary centre of mass. On the other hand, they have constructed a system where such is (almost) the case.

Neither Nicola or Tim took school physics to GCSE level and this showed in their confusion of terms such as position, velocity, force and gravity. Although the opportunity was not taken in the study, Gravitas does provide an environment for correcting these kinds of misconception, in the context of systems which the subjects have constructed themselves.

3.2.4 *Creating a Massob System: Transcript 2*

Our second transcript is of an 18 year old school student who specialised in physics and mathematics at A level. This subject has a strong interest in astronomy and he already knew the radius of the Earth and the distance to the Moon. Because of this he found the construction of the Earth and Moon Massobs and the setting of their values quite straightforward. He was also quite familiar with Conservation of Momentum, within the contexts treated in school physics. Nevertheless, he was quite surprised when he noticed the orbital procession. His efforts to explain the motion to himself were successful, and he went on to discover how he could modify the initial conditions in such a way that the centre of mass remained stationary. It is the transcript of his explanation and discovery that we will concentrate on here.

The subject, Simon, has already constructed the Earth and Moon Massobs and has noticed that the system as a whole moves down the screen:

R: "Can you think why they are moving down like that?" [Figure 3.12]

S: "You've actually got it so that the gravity of the Moon is affecting the Earth?"

R: "That's certainly true. *All* the gravitational forces are computed by Gravitas."

S: "There's nothing else causing it to move?"

R: "No."

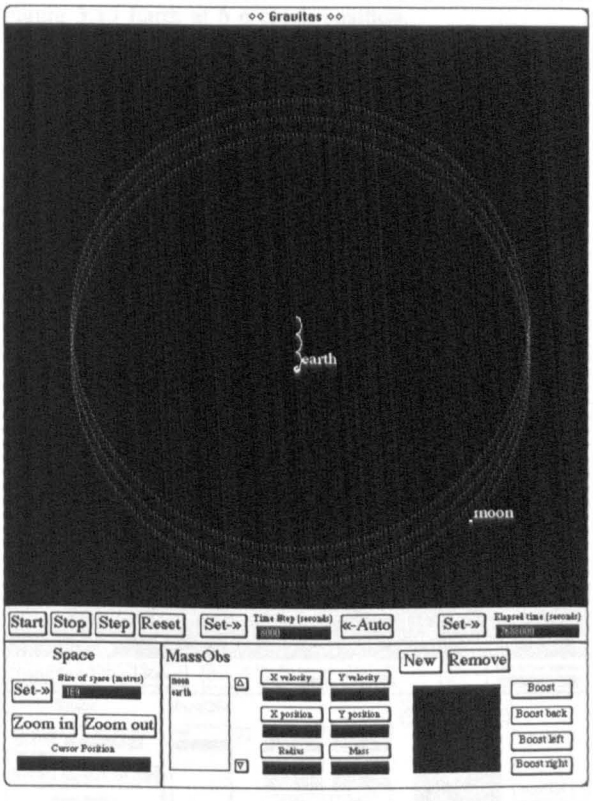


Figure 3.12 Simon's encounter with Orbital Procession

R: "I'll talk you through it and get you to explain it to your own satisfaction. What are the initial conditions, of the Earth?"

S: "Zero."

R: "And the Moon?"

S: "Minus 1000."

R: "Let the Moon go from 3 o'clock to 6... What will the Moon have been doing to the Earth?" [Figure 3.13]

S: "It will have pulled the Earth by an amount equal to the ratio of the masses."

R: "What has it done to Earth's velocity?"

S: "Increased it."

R: "Take the Moon to 9 o'clock. What will that have done?"

S: "I'm not sure exactly. The across velocity from 3 to 6 will have been cancelled out?"

R: "Right, what about the pull on the Earth during the whole of that semi-circle?"

S: "It'll have given it a downwards velocity. The x-component is balanced out." [Looks at Earth's values on the displays and sees that the x-velocity is indeed very small. Figure 3.14

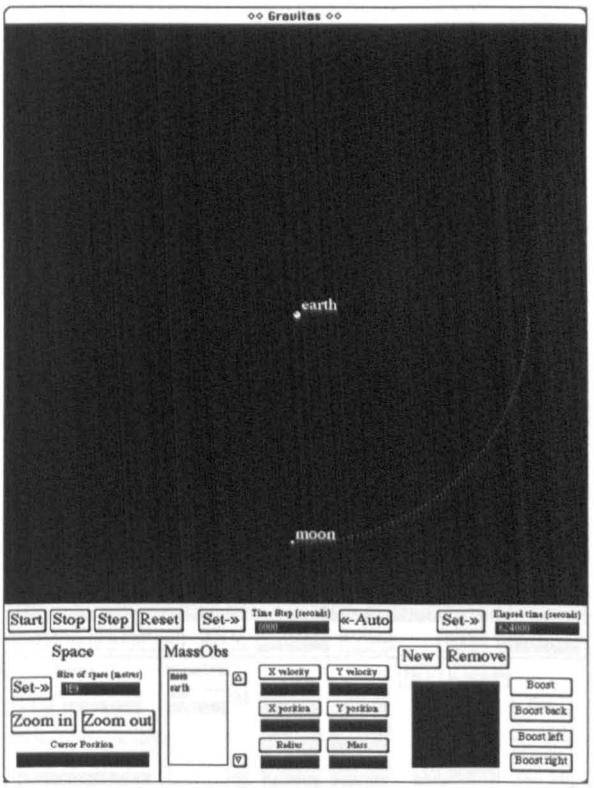


Figure 3.13 Earth at 6 o'clock position.

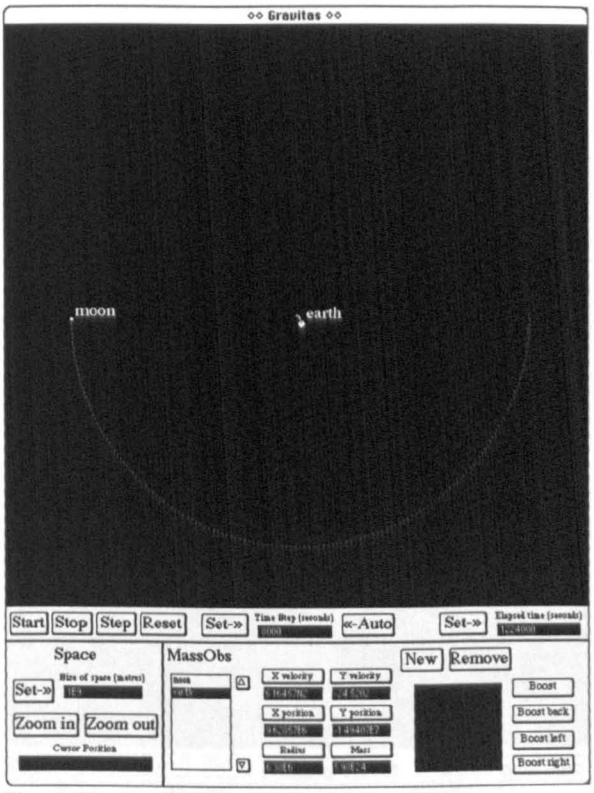


Figure 3.14 Earth's x-velocity close to zero

S: [Takes Moon to 12 o'clock] "This will have the effect of slowing it but giving it greater negative x."

S: [Takes Moon to 3 o'clock] "This should reduce the y to zero and take it back to rest. Well, sort of rest anyway." [He checks the values and finds the Earth is almost stationary. Figure 3.15]

R: "Good. You've just talked through the forces that have acted on the Earth... But now think about the initial conditions again. What velocities are there in the system?"

S: "Zero and 1000."

R: "What could we do so that we got things running on the spot?"

S: "Spin the Earth?"

R: "We can't have spin."

S: "It wouldn't work would it? Giving the Earth a sort of negative momentum? Opposite to the one you've got for the Moon?"

R: "How about trying that?"

S: "I need to figure out the values. It's 1000 and the mass [of Moon] is that, so it needs to be 7.353E25 divided by that [mass of Earth], 5.98E24..."

He uses the calculator and obtains 12.296. Sets Earth's velocity and starts the system. The orbit no longer processes down the screen. Figure 3.16.

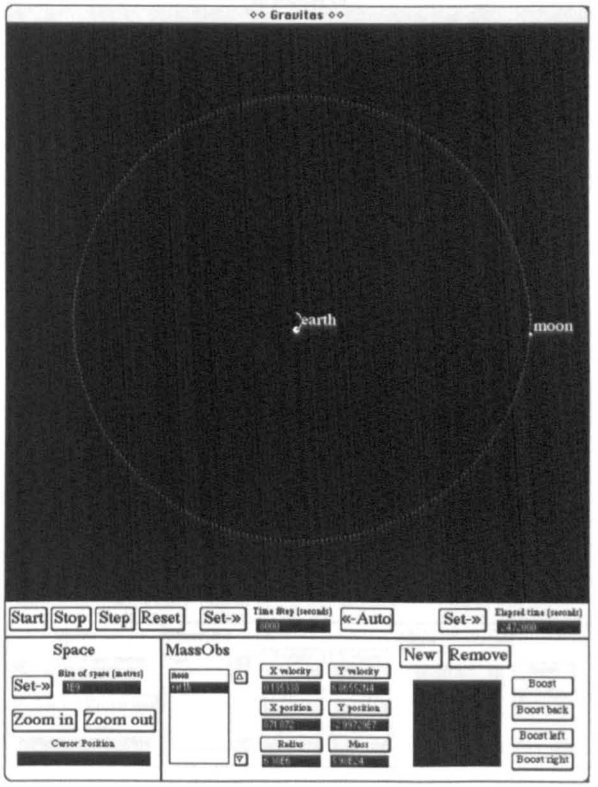


Figure 3.15 Earth almost stationary

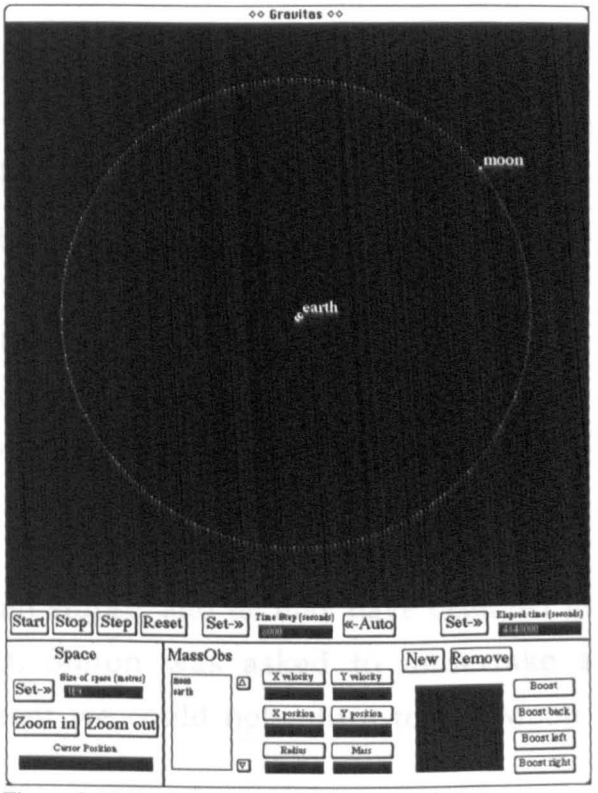


Figure 3.16 Earth orbiting the Centre of Mass

R: "Why not try zooming right in..." [Figure 3.17]

S: "Oh, it's a circle..."

R: "And what is interesting about the centre of that circle?"

S: "It's the centre of mass."

R: "Yes, and it is beneath the surface of the Earth."

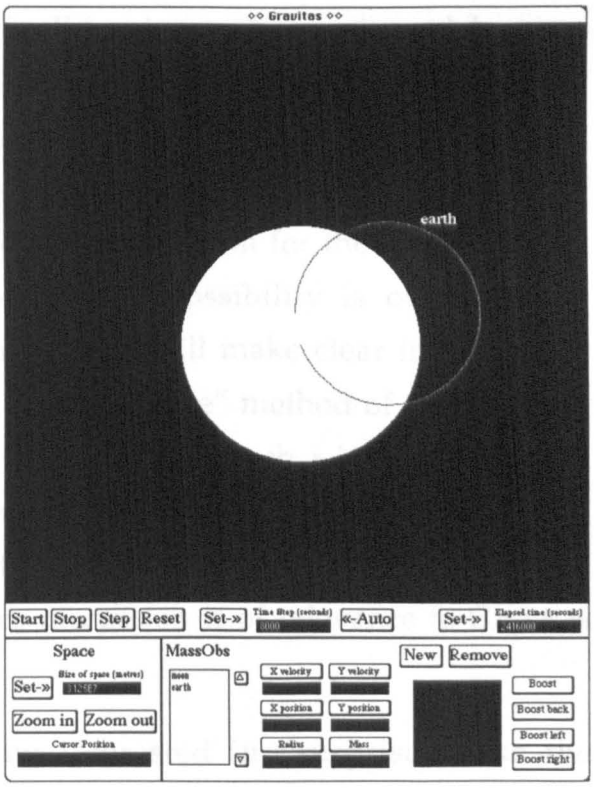


Figure 3.17 Close-Up of Earth orbiting Centre of Mass

3.2.5 *Discussion*

Simon spent just over forty minutes on the sequence described above. As the transcript shows, he found it quite easy to think his way through the Orbital Procession Surprise even though he was not expecting it. His intuition about giving the Earth a momentum equal and opposite to the Moon to create a stationary system was swift and accurate, suggesting that he already had a good grasp of the formal concept. In this instance, Gravitas was giving Simon the opportunity to apply his prior knowledge to construct the solution of a problem. Examining the transcript now, it seems regrettable that the line of investigation was not continued, to examine Simon's understanding of the physics more deeply. For example, the path of the Earth around the centre of mass is not actually a perfect circle but an ellipse, and he may have been led to discover this. However, in the event, Simon was asked to undertake a different task, and time restrictions meant we could not return to the Moon-Earth system.

3.2.6 *A Second Look at Surprises*

We have shown that both scientifically naive and experienced learners can construct simple gravitating systems and then be surprised by their behaviour. What Gravitas provides is a new medium for thinking about and playing with such situations. One of our hopes for Gravitas is that this way of looking at physical problems can be a useful preparation for the more abstract, mathematical treatments. Investigation of this possibility is obviously an important line of research for the future, as we will make clear in chapter 7, but there are also grounds for believing this "concrete" method of dealing with difficult concepts has its own intrinsic worth. In fact, such a belief has been a common thread in Papert's work and he has recently made a strong plea that educators should assign equal value to both concrete *and* formal "ways of knowing" (Turkle and Papert, 1990). This discussion is of relevance to Gravitas and we will return to it in chapter 6.

We have another reason for being interested in "surprises" like the orbital procession described in the dialogue above. In the early days of Turtle Geometry researchers discovered that even quite simple programs could cause the Turtle to produce strange and beautiful patterns. Abelson and diSessa (1980, p20) refer to the "surprising" behaviour of a small procedure called **inspi**. Figure 3.15 gives two illustrations of this procedure (translated from Abelson and diSessa's notation into modern Logo).

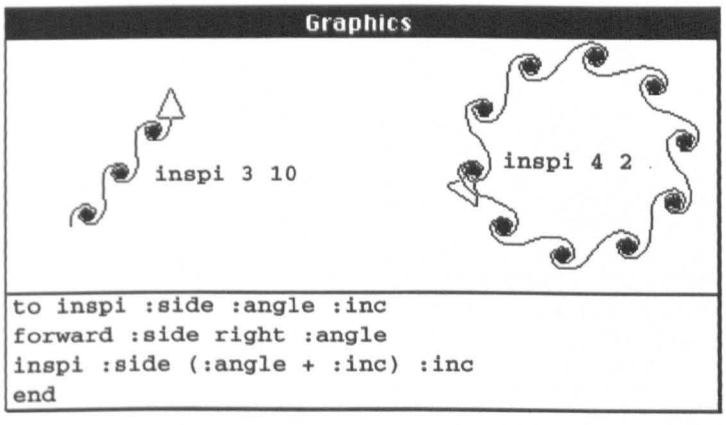


Figure 3.18 The **inspi** procedure (Abelson and diSessa, 1980, p20)

Far from being just trivial squiggles on the screen, patterns like those in the figure are used by Abelson and diSessa as the gateway to a host of theorems about the mathematics of Turtle Geometry. In fact, Abelson and diSessa's book presents a substantial mathematics curriculum in which conventional proofs and lemmas are mixed with innovative procedural demonstrations (Logo

procedures, that is) of important theorems. In view of this, we wondered if Gravitas could exhibit analogous surprises which could form a basis for investigations into physics with Massobs. The orbital progression surprise, and others we have found, show that it can. The development of a programme of Massob centred physics, comparable to the mathematics curriculum of Abelson and diSessa's book, is beyond the scope of this thesis. We have demonstrated, however, that Gravitas does have the potential to support such a project.

3.3 *Constructing a Program*

3.3.1 *The Task*

As we noted in chapter 2, Gravitas did not originally have a graphical interface. This was added later, with the intention of making Gravitas easier to use. We expected that learners would at first rely upon the graphical interface and then, as they became used to the system, carry out more of their work through programming, leaving the buttons behind. A similar progression is often encouraged in Turtle Geometry: children are at first taught to drive the Turtle around the screen using single Logo commands or "instant keys" but they soon advance to write programs which create more complex drawings. However, with Gravitas we discovered that although the amount of programming did rise with familiarity, there was little or no reduction in the use of the graphical interface. In fact there seemed to be a synergy between the two interfaces which allowed the students to take on more complex projects than would otherwise have been the case.

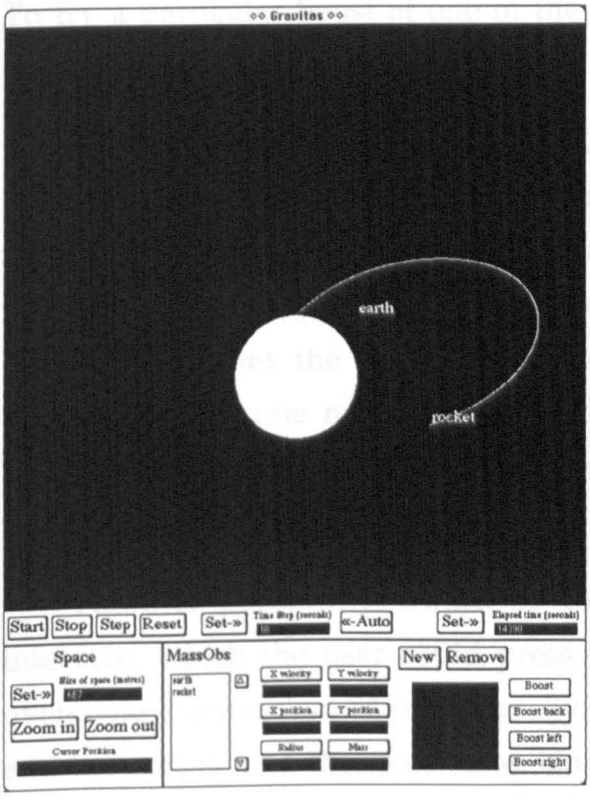


Figure 3.19a A Rocket falling back to Earth

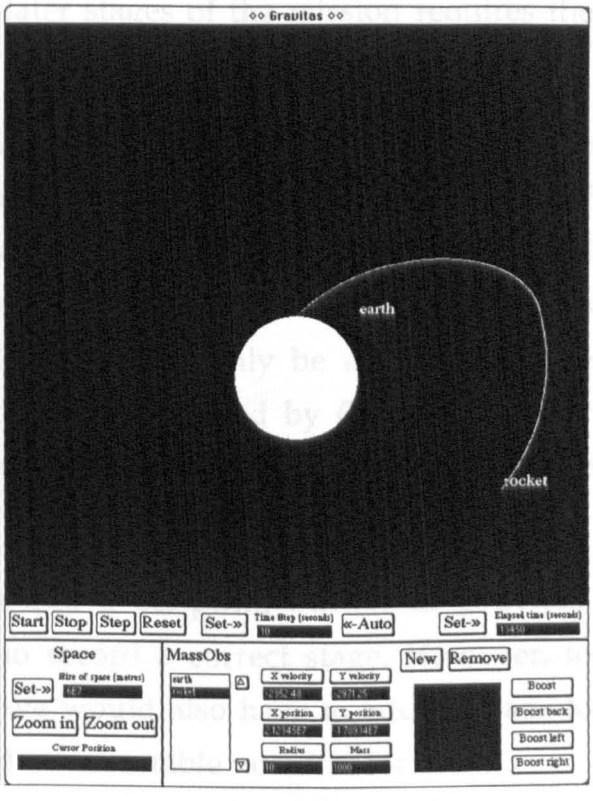


Figure 3.19b A Rocket boosted at apogee

To examine how learners could actually exploit this synergy, we decided to invent a task which would be difficult to perform without using both interfaces. In an early study we had observed users launching a "rocket" Massob and boosting it into orbit around a Massob representing the Earth. This

task can be carried out using just the graphical interface. First the two Massobs are created and given appropriate masses, then the rocket is given a velocity which takes it to a substantial height above the Earth (Figure 3.19a). Finally the user must discover that to achieve circular orbit a specific boost must be given to the rocket at its apogee (Figure 3.19b).

We realised that a sequence of such manoeuvres could be combined to form a journey to the Moon and back, similar to the Apollo missions of the late 1960s. However, we also saw that the construction of the sequence would be difficult using the graphical interface alone. The difficulty arises because each *new* stage in the journey must be built onto a "debugged" prior stage. A boost which sends the rocket towards the Moon must be preceded by a boost which successfully sets the rocket into circular orbit around the Earth. In turn this must be preceded by a successful launch. The boosts which initiate each stage of the journey must be accurate in their strength and timing, and as there are no straightforward mathematical formulae to help plan such a mission, the easiest way for learners to determine the correct values is by trial and error. To try a particular boost at one of the later stages of the mission requires the learner to "replay" each of the previous stages correctly.

There are several ways around this problem. With Gravitas in its present version the user can save a system to disk at any point, so a sequence of debugged stages could be created, saved, and finally replayed. A critical drawback to this approach is that the construction of the mission is not always linear. Sometimes the success of a late stage can only be achieved by the modification of one much earlier (a possibility offered by Gravitas but *not* available to the Apollo astronauts) which would force the user into some contorted file handling.

Another possibility would be to add a scripting button to Gravitas' interface, which the user could press to record a correct stage. However, to overcome the problem of non-linearity we would also have to add a means to edit the scripts, and therefore some kind of executable notation for them.

The clearest and most natural solution is to write a program to control Gravitas via the programming interface. The programming medium is standard Logo and debugged stages can be encapsulated in procedures. Used in tandem, the graphical interface allows the rapid testing of varied boost strength and timing, while the repeatability of the programming interface allows the

overall solution to be approached incrementally. The user can incorporate zooms into the program to control the detail of what is visible on the screen, and the time step may be manipulated to control the rate at which the mission replays. At the end, the user is left with a single definite item, a program, which controls Gravitas to produce a moving picture of the mission. Later, the user can return to the program and modify or extend it. The transcript we present below is a detailed record of two subjects creating such a program, using the graphical interface at critical points.

3.3.2 *A Mission to the Moon: Transcript 3*

Joe and Dan are 13 year olds who take a broad range of subjects at school and have not yet specialised in science or the humanities. Although they do not use computers at school, Joe and Dan both have home computers and are familiar with keyboards, mice, and graphical interfaces. However, neither of them had any prior experience of Logo or programming. Nevertheless, they took only a few minutes to learn how to operate Gravitas. As with the two studies described above, they began work by constructing Earth and Moon Massobs and discovering the correct orbital conditions. As usual, they were surprised by the orbital procession, but eventually explained it to themselves satisfactorily once the researcher had led them to consider the forces acting over each quarter revolution. They were then asked to begin the task described above: launch a rocket from the Earth and fly it out to orbit the Moon.

Although they have no particular interest in astronomy, Joe and Dan were aware of some relevant details of the Apollo missions to the Moon. For instance, they knew that the journey to the Moon started from an orbit around the Earth rather than directly from a ground based launch. Accordingly, they decided to create a rocket, place it at the twelve o'clock position on the Earth's surface, and experiment with different values for its launch velocity. Using the buttons of the graphical interface they set the mass and radius of the rocket to reasonable values (1000 kilograms and 10 metres) and began to think about values for the x and y velocity components.

After a short discussion they decided to give the rocket equal velocity components because then it would "go at 45 degrees". We begin the transcript at this point. Figure 3.20 shows the rocket having been launched with x and y velocities of 7000 metres per second.

They press **Start**. As the rocket travels away from the Earth the researcher asks what they imagine will happen.

R: "What's it going to do?"

Joe: [as the rocket passes Point A of figure 3.20] "Whiplash round the Earth." [He indicates an orbit]

J: [as the rocket passes Point B] "It's going to crash back to Earth."

R: "Which?"

Dan: [at Point C] "We'll go for crash back to Earth."

The rocket falls back to Earth.

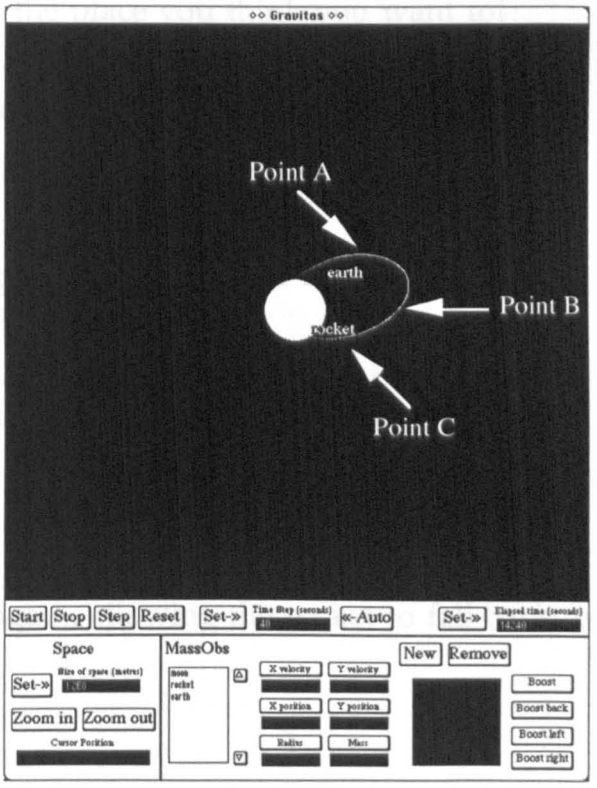


Figure 3.20 Launching the rocket.

R: "Have you any plans for how to get it to orbit the Earth?"

J&D(together): "Boost it!"

They **Reset** the system and **Start** again. After the rocket has travelled some distance away from the Earth they **Boost** it. The rocket still falls back to Earth, but in a different place. Figure 3.21.

J: "That boost has made it go more wide, hasn't it."

R: "Where do you think is the best place to boost it to get it into orbit?"

J (to D.): "Well, we boosted the Moon right? Well, set it going and it was in orbit, so you boost the orbit when it's in the right flight path. [he describes a tangent to the apogee of the rocket's flight]

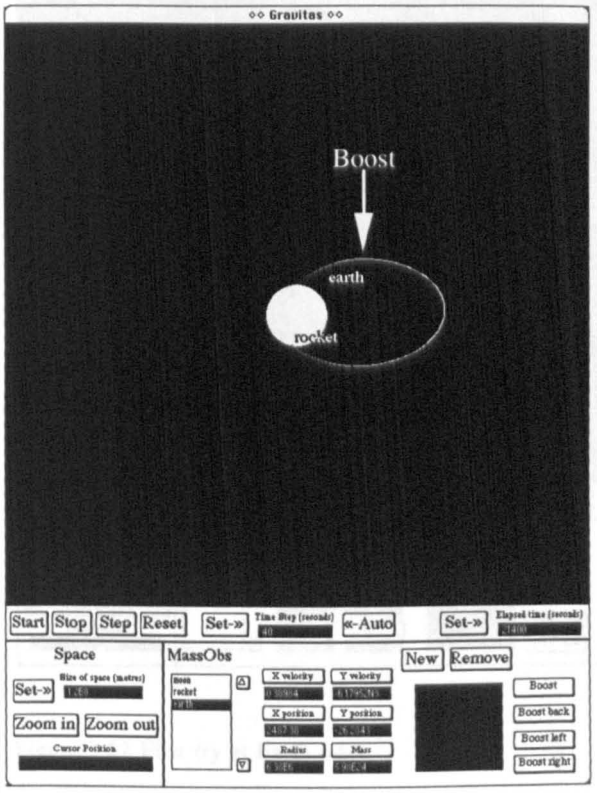


Figure 3.21 **Boost**ing the rocket.

R: "OK. Start again and point to the place you think you want to boost it."

J: "I want to boost it when it's running parallel to the Earth."

R: "Right, how would you find out when that point is? Have you got any smart ideas?"

J: [Points roughly at the right spot] "Should do all the boosts at once, when it's parallel to Earth."

R: "There's a clever way to find out when you should boost it. I'll give you a clue: you should launch it and let it crash back to Earth."

They **Reset** the system and **Start** again, allowing the rocket to fall back to Earth, with no extra boosts.

R: "How long did that take?"

J: [Examines the elapsed time counter] "14,240 seconds."

D: "Half that!"

J: "It's like you're throwing a ball up. You should get a peak halfway"

They **Reset** the system and **Step** it until 7,120 seconds have passed. Then they **Boost** the rocket 6 times. Joe has already realised this is a way to stack several boosts at a single instant. Figure 3.22.

R: "I want it to be the same distance from Earth all the way round."

D: "Shall we do the same beginning move again?"

J: "Yes."

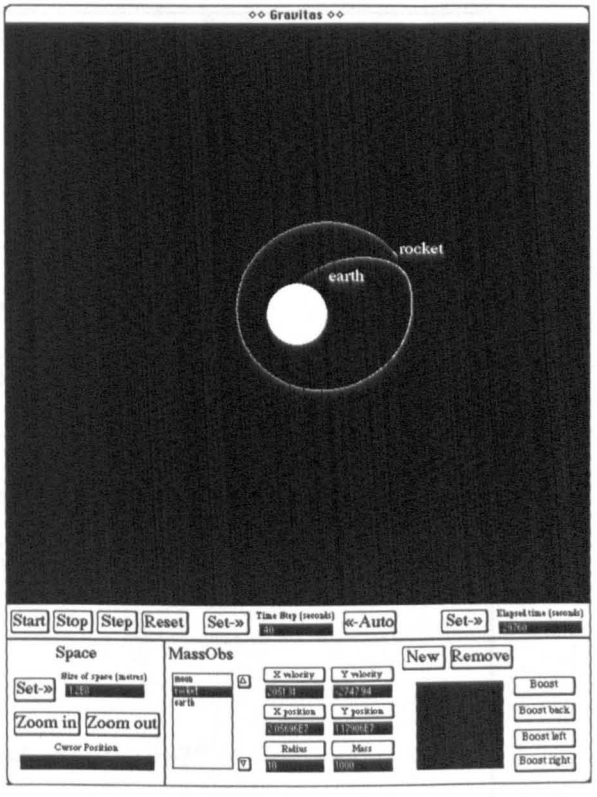


Figure 3.22 First try at Earth orbit.

Still using the buttons, they **Reset** the system and **Step** it until 7,120 seconds have passed. Then they **Boost** the rocket 6 times, **Start** the system and allow it to run until the rocket comes back around to the same point and **Boost** it 3 more times. Figure 3.23.

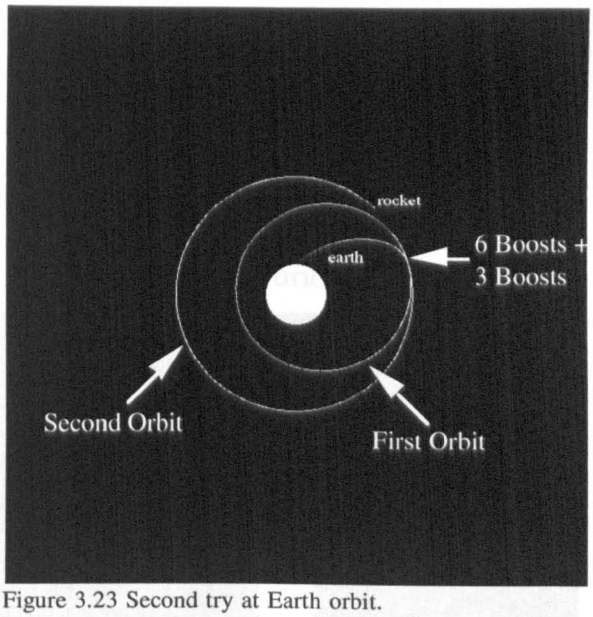


Figure 3.23 Second try at Earth orbit.

D: "That's it."

R: "That's good enough. OK, how would you get it to do that straight off?"

J: "Boost it 9 times."

R: "Yes."

They have obtained a satisfactory orbit for the rocket, using the buttons. The researcher now guides them to write a program to do the same job:

R: "Right, now I'd like you to write a program."

D: "Alright then, go on."

J: "To do what?"

R: "To do this, instead of using buttons all the time."

There is a program editor alongside Gravitas:

R: "Try typing `reset` in that window and then pressing Command-R."

They do it and the system resets just as if they had pressed the button.

R: "Can you guess any other commands?"

D: "We can't use Step can we?"

J: "Boost?"

D: "No, we want it to get there first, don't we?"

J: "Yeah."

R: "So you want it to go until..."

J: "Right, until 7,120 seconds."

R: "Yes, right, there's a command called `go.until.time` and after it you put a number and it will go until that number of seconds." [explains that `go.until.time` is all one word]

They type **go.until.time 7120** after the reset and run the program to produce a launch. The system is running slowly however, with an animation step of 20 seconds and they ask if they can speed it up:

R: "You can use **set.time.step**, followed by the seconds."

They now have a three line program for the rocket launch:

```
reset
set.time.step  40
go.until.time  7120
```

They run it. Figure 3.24.

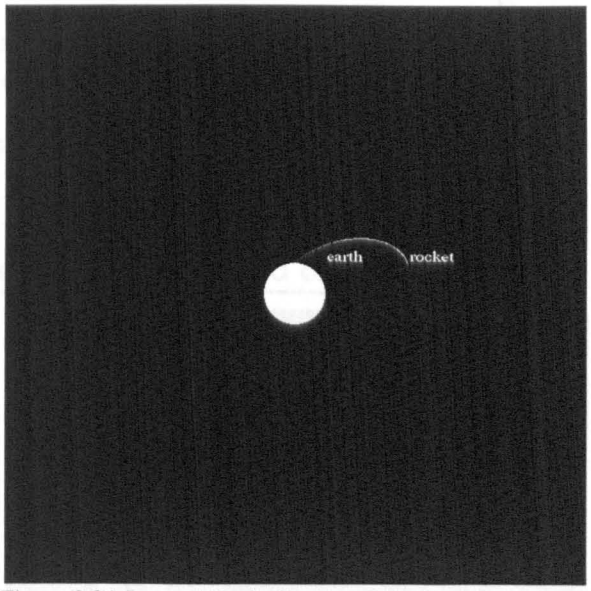


Figure 3.24 Joe and Dan's first program.

D: "We want to boost it. Is it just like boost?"

R: "It's just boost, but you have to put the name of the object you want to boost."

J: "Boost rocket? Just do it nine times?"

R: "Well, there's another command called repeat..." [Explains how **repeat** works]

R: "...then you put what you want to do inside square brackets."

J: "Boost rocket."

R: "I'm afraid you have to put a colon before rocket in there." [They don't ask why!]

They run it and get a launch followed by nine boosts. Then nothing happens.

R: "What do you need now?"

J: "Start?"

R: "Actually, it's **start.animation**."

Their program is shown below. They run it. Figure 3.25.

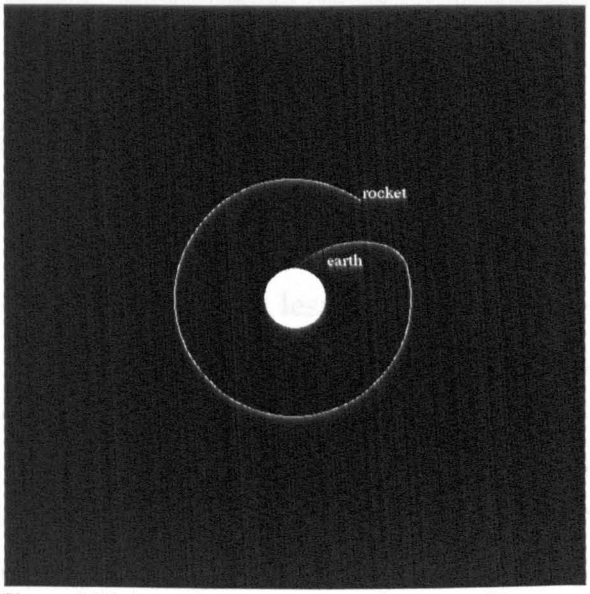


Figure 3.25 A program to get the rocket into orbit.

```
reset
set.time.step  40
go.until.time  7120
repeat  9  [boost  :rocket]
start.animation
```

R: "I think that's a great success."

D: "Yes, it's a circle..."

J: "It's following that line exactly." [The rocket is moving along its second orbit]

R: "We better save your program." [Shows them how to save the program to disk]

R: "Next I'd like you to get the rocket to go out to the Moon and orbit it. How are you going to do it?"

J: "Boost out. Scan out so we can see the moon."

R: "When?"

J: "We want to boost out so powerful that we get rid of the pull."

D: "Yes, but it's going to do a circle and actually join that." [Indicates a realistic transfer orbit to the Moon]

J: [Quickly agrees with Dan's plan] "We want to boost it here." [He points to a boost where the rocket is opposite the Moon with respect to the Earth]

Figure 3.26. This is quite surprising. They have both seized on the idea of a semi-circular transfer orbit rather than the simpler (but ultimately incorrect) direct boost at the moon.

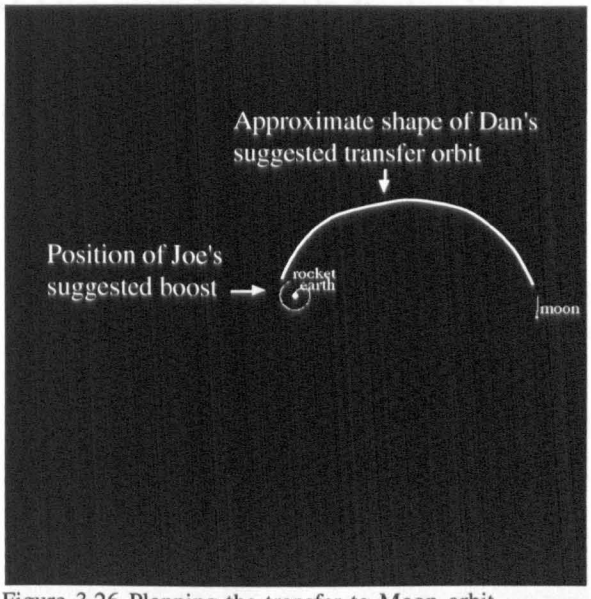


Figure 3.26 Planning the transfer to Moon orbit.

The semi-circular transfer orbit is indeed the method used in Moon shots (Baker, 1967). This is because it is relatively insensitive to boost errors and is the most fuel efficient way to travel between two bodies. However, it is somewhat surprising that our subjects should follow this less obvious "Apollo mission" method, and not the ballistic or "Jules Verne" tactic of heading straight for the Moon (In his novel, *From the Earth to the Moon*, the lunar vehicle is fired from a huge gun aimed ahead of the Moon). It would perhaps have been interesting to ask the subjects how they got the idea for the transfer orbit, but at the time the researcher thought this might interfere with the task. In other studies of the same task, two subjects chose the same course, both because of their knowledge of the Apollo missions. Another pair had to be steered towards the Apollo method after trying a direct shot with no success.

R: "OK. Run the program and look for the time to boost."

D: "If we make it go up here we need quite a few more boosts." [He points at the transfer orbit again]

J: "We don't want to boost it too much because as it breaks away it gets less pull."

D: "Will the Moon pull it?"

R: "The Moon will pull it, but only when it gets really close."

J: "We've got a perfect circle here, so we only want it to break out of it because once it's out it should accelerate away." [Indicates the transfer orbit again]

D: [to J.] "Where do you mean? Like over there? [Indicates 11 o'clock]

J: "Just slightly diagonally." [Points at about 1 o'clock]

Joe decides to boost at 35,000 seconds.

R: "How many boosts?"

J: "We don't want to boost it too much or it will just break away."

D: "Four."

Using the buttons, they **Step** the system to 35,000 seconds, then **Boost** the rocket 4 times and observe the results. Figure 3.27.

The boost is clearly insufficient to reach the Moon but Joe and Dan are more concerned with the aiming, which they feel is faulty:

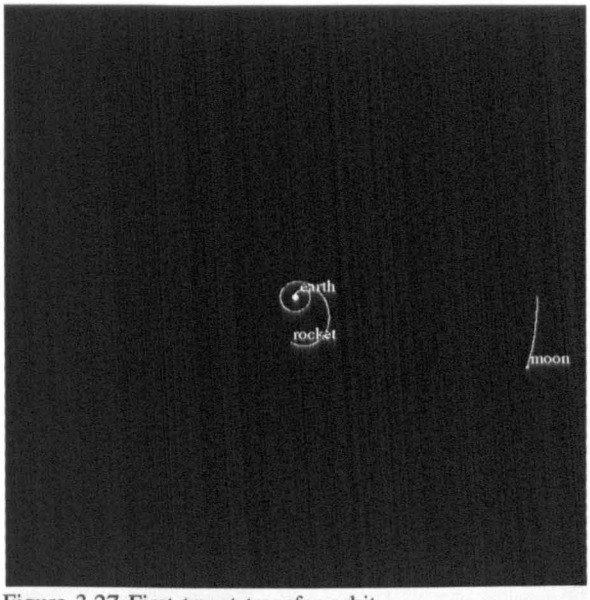


Figure 3.27 First try at transfer orbit.

D: "Remember, I said you had to do it *there* [he points at 11 o'clock again] because then it will come round."

R: "Earlier or later?"

D: "Earlier."

J: "32,000 seconds."

This time they add the new boost time to the program and run it. Figure 3.28.

```
reset
set.time.step  40
go.until.time  7120
repeat  9  [boost  :rocket]
go.until.time  32000
repeat  4  [boost  :rocket]
start.animation
```

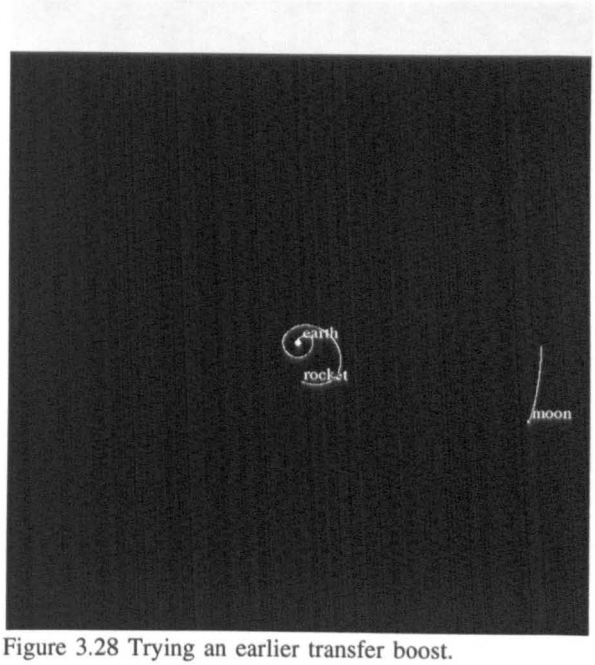


Figure 3.28 Trying an earlier transfer boost.

D: "The Moon's moving, it's not enough."

J: "Shall we change the boosts? Or shall we concentrate on where we do it?"

D: "Give it 10."

They edit the program to 10 boosts at 32,000 seconds and run it. The rocket flies out of the system, almost without deflecting. Figure 3.29.

J: "It's going to break out."

D: "Change the program to 6 boosts."

Joe does this and also adds a line to increase the time step so that the system runs faster.

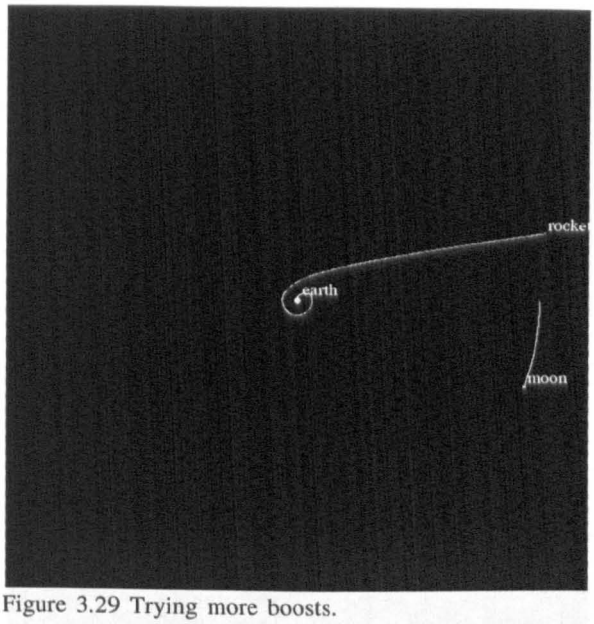


Figure 3.29 Trying more boosts.

D: "That's good."

J: "Not far off, that."

Figure 3.30. Using a combination of button presses and program alterations they have got quite close to the Moon. The actual transfer orbit is rotated compared to their original plan because of the orbital motion of the Moon but it can be seen that the concept was sound. They now have to fine tune the transfer orbit. Their program reads as follows:

```
reset
set.time.step  40
go.until.time  7120
repeat  9  [boost  :rocket]
set.time.step  200
go.until.time  32000
repeat  6  [boost  :rocket]
start.animation
```

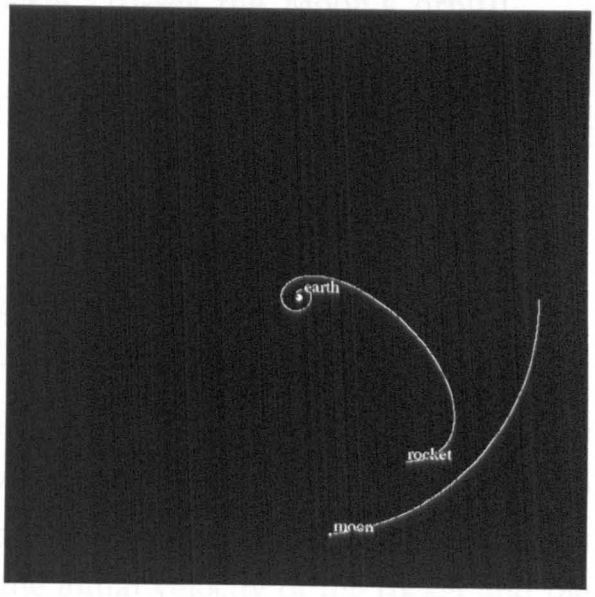


Figure 3.30 6 boosts at 32,000 seconds.

J: "Let's make it 34,000."

D: "Try 7 boosts."

The rocket passes very close to the Moon. As it gets close they use the **Step** button to take over from program control and watch what happens in detail. Figure 3.31:

J: "Do we want to meet it [the Moon] on the inside or the outside?"

R: "The Apollo missions met it on the outside."

D: "Shall we boost back?"

J: "No, just step it because we don't know what will happen."

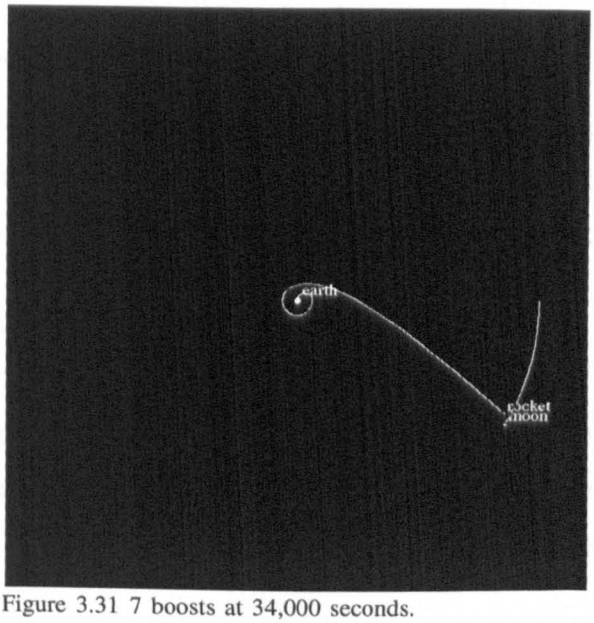


Figure 3.31 7 boosts at 34,000 seconds.

They now decide it is tiresome running the program from the initial positions each time, and as each run takes about one and a half minutes they have a point.

J: "Can we set it so we carry on from that point? [indicates a position just before the rocket crosses the Moon's orbit] About 216,000 seconds?"

R: "Yes. Just replay it and save the system at that point."

Having done this, they are in a position to do quick trials with the buttons, around the point of interception.

J: "To orbit the Earth we need 9 boosts, so we need about 1, so we need to take it back 15 boosts."

Joe's line of thought seems to be that 9 boosts got the rocket to orbit the Earth; the Moon is much smaller so it will need say 1 boost; there have been 16 boosts altogether so far, therefore giving it 15 back boosts will do the job. This is quite ingenious, but flawed: It neglects the initial velocity of the rocket and the energy used up climbing out of Earth's gravity well. Dan does not agree:

D: "Try about 10 back boosts."

J: "To work it out accurately, you'd need to know the gravitational equivalent of a boost."

They press the **Boost back** button 10 times and **Start** the system.

J: "Oh no, it's stopped dead... It's falling back to Earth."

D: "Try 5 back boosts."

Dan also begins to rethink the timing of the transfer boost. This is an example of the non-linearity which can arise in the construction of programs: The subjects were concentrating on braking at the Moon but have skipped back to an earlier stage:

D: "I say we do it at 35,000."

This time they modify the program rather than press buttons. They alter the transfer boost timing, add `go.until.time 216,000` and give the rocket 5 back boosts (having asked the researcher for the name of the command):

Their program now reads:

```
reset
set.time.step  40
go.until.time  7120
repeat  9  [boost  :rocket]
set.time.step  200
go.until.time  35000
repeat  7  [boost  :rocket]
go.until.time  216000
repeat  5  [boost.back  :rocket]
start.animation
```

They run the program. Figure 3.32. The figure is actually the state of affairs at 436,000 seconds. The rocket passes *in front* of the Moon and carries on a short distance outside the Moon's orbit. It starts to fall back to the Earth after picking up some momentum from the Moon which causes the small loop. Joe and Dan realise this is not *entirely* wrong:

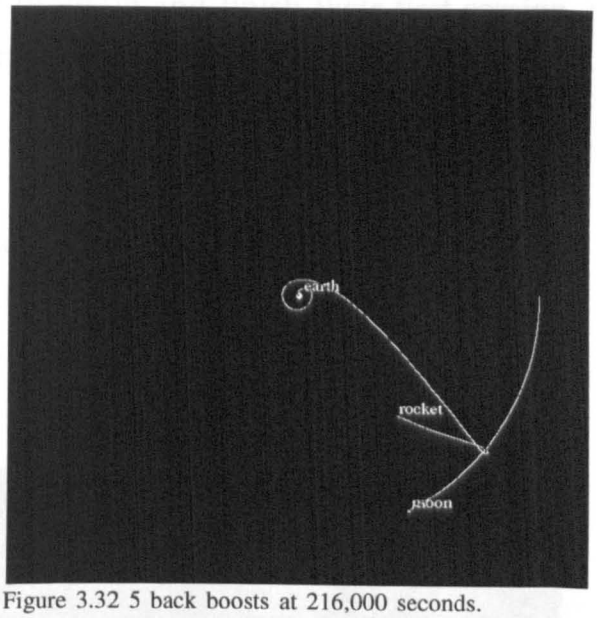


Figure 3.32 5 back boosts at 216,000 seconds.

J: "Back boosts are OK though. I'm going to try 34,500."

Again, they modify the program.

J: "We've landed on the Moon!" [Figure 3.33]

R: "At about 10 times the speed of sound!"

D: "You need to drop the go.until.time."

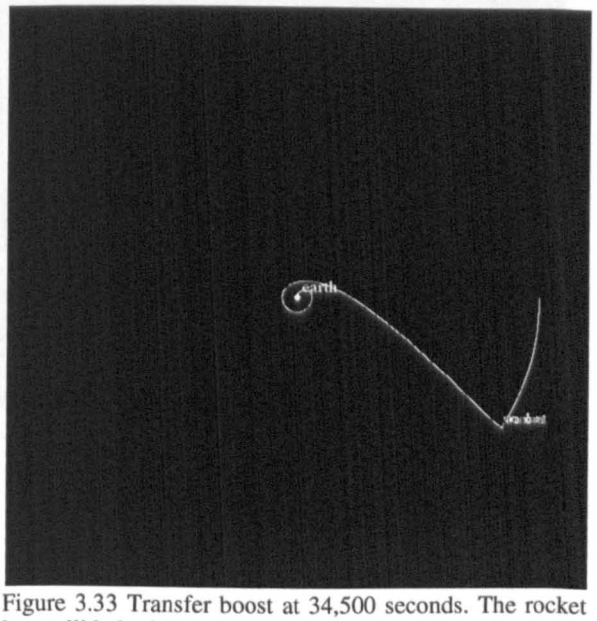


Figure 3.33 Transfer boost at 34,500 seconds. The rocket has collided with the Moon (overwriting their names).

Joe edits the program so that the back boosts occur at 214,000 seconds. The rocket passes very close to the Moon, but has too much residual velocity and flies off. Figure 3.34.

J: "That's good. We need another back boost just after rendezvous."

R: "That's a good word for it. The correct word."

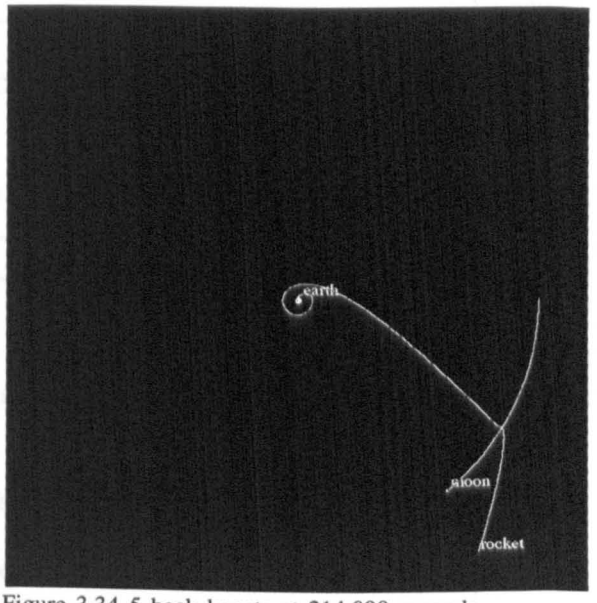


Figure 3.34 5 back boosts at 214,000 seconds.

At this point Joe and Dan save their program and finish their first session with Gravitas, which has lasted just under two hours. Before leaving they make a short note to themselves for the next session: "Try a back boost after 216,000."

At the beginning of their next session, 5 days later, they read their note and run the program to remind themselves of where they had got to. They start to make more changes to their program:

J: "Yes, we need to back boost once more, just after there." [he indicates a point just outside the Moon's orbit]

Joe **Steps** the system through the rendezvous point and decides to apply the back boost at 230,720 seconds.

J: "How many back boosts shall we try?"

D: "Two?"

Joe presses the **Boost back** button twice and then **Start**s the system again. However, the rocket crashes into the Moon at 287,000 seconds. Figure 3.35.

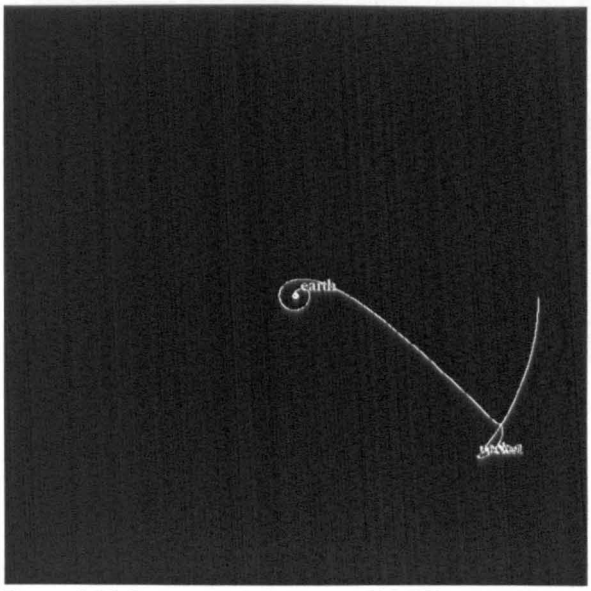


Figure 3.35 2 more back boosts at 230,720 seconds.

D: "Try a boost there."

Dan indicates a point where the rocket is travelling next to the Moon, before it crashes. Still using the buttons, Joe tries a **Boost** at 260,000 seconds. The rocket still hits the Moon, but quite a bit further on.

D: "It's too slow. We need a boost at the same time."

Joe replays the system and **Boost**s the rocket twice at 260,000. Now the rocket escapes again. Joe and Dan conclude that 1 boost is too few and 2 is too many. The researcher explains that they can control the boost strength with a new command: `set.boost.strength`.

R: "The boost now is 250 metres per second."

D: "Let's try 5 times 105"

This does not work either. Dan has miscalculated: 5 times 105 is more than 2 times 250. He realises this and they decide to try 4 boosts at 100 metres per second. Joe wants to alter the timing as well:

J: "I reckon we've got to change it here."

Joe edits the program which now reads:

```
reset
set.time.step  40
go.until.time  7120
repeat  9  [boost  :rocket]
set.time.step  200
go.until.time  34500
repeat  7  [boost  :rocket]
go.until.time  214000
repeat  5  [boost.back  :rocket]
go.until.time  230720
repeat  2  [boost.back  :rocket]
go.until.time  250000
set.boost.strength  100
repeat  4  [boost  :rocket]
start.animation
```

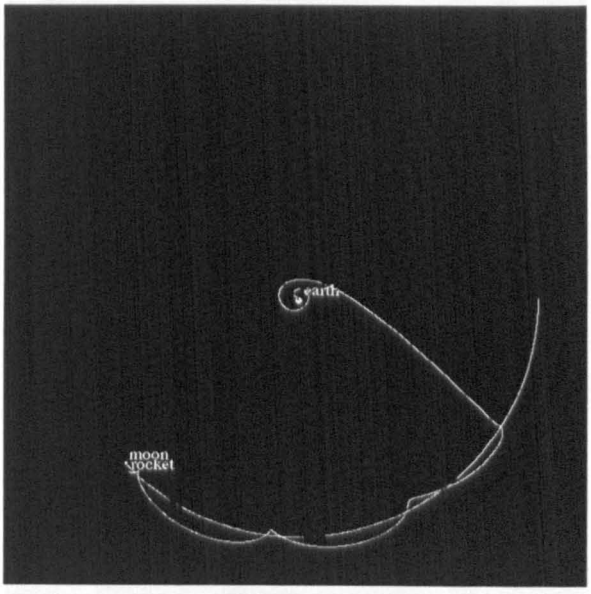


Figure 3.36 4 boosts at 250,000 seconds. The rocket is orbiting the Moon.

D: "That's it."

R: "Congratulations. You have got the rocket to the Moon. Now, can you get it back again?"

First of all though, they try to get the rocket to orbit a little closer to the Moon. They experiment, using the buttons, with back boosts as the rocket is overtaking the Moon and settle on a single back boost at 686,720 seconds. The period of the rocket's orbit around the Moon reduces slightly. Figure 3.37.

D: "It will be like a flower soon."

They return to the task of getting the rocket back to Earth.

J: "You know those peaks? If we get on one of those and just shoot back to Earth..."

This is pretty much the right idea. It is not too surprising that Joe should guess this straight away: As one watches the system running there are times when the rocket clearly seems to be heading towards Earth. It is intuitively appealing to boost at this point.

They replay the system looking for the time at which the "peaks" occur and choose 1,137,520 seconds.

J: "OK. 6 boosts at 1,137,520 seconds"

They run the program from the beginning again and watch as it does indeed fall back to Earth:

J: "The Earth's going to get a really good hold of it in a minute"

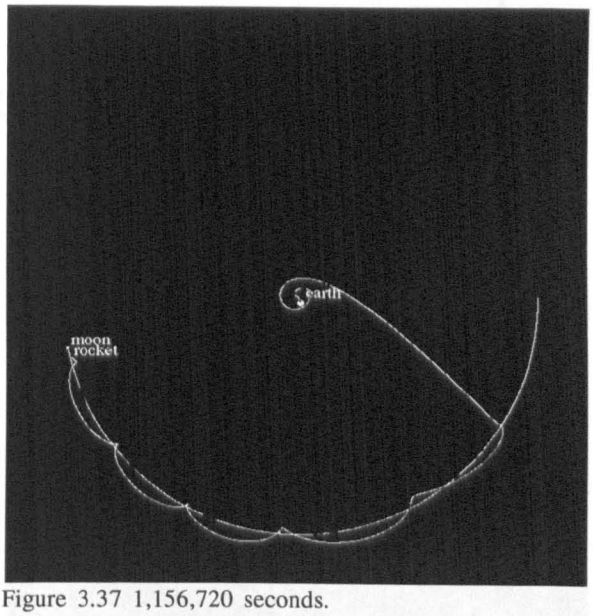


Figure 3.37 1,156,720 seconds.

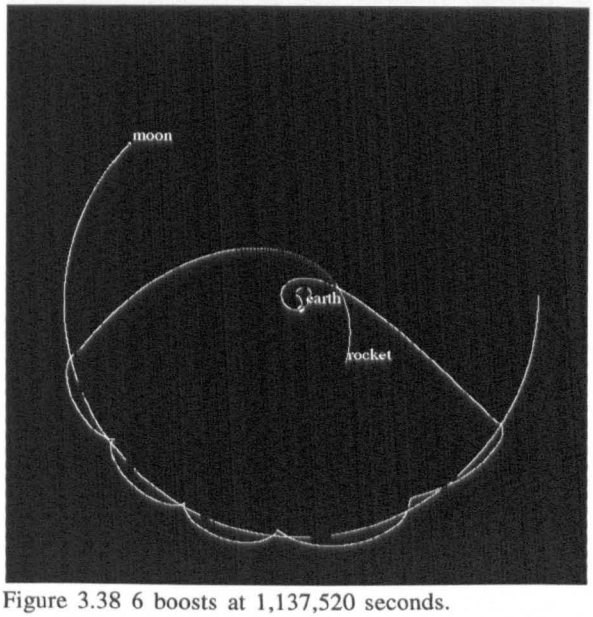


Figure 3.38 6 boosts at 1,137,520 seconds.

However, the rocket has gained a large velocity and travels past the Earth (Figure 3.38). Joe and Dan have two ideas: First reduce to 5 the number of boosts at the "peak" and then to brake the rocket with back boosts as it gets near the Earth. Using the buttons they experiment to find a suitable time for the back boosts:

D: "Boost it back at 1,497,520." [He reads the time display]

J: "How many?"

R: "How many did it take you to get away from the Earth?"

J: "Yes, 9. Let's do 8."

D: "That's less than last time. Do 10."

Joe modifies the program and runs it. The rocket travels back and brakes into an orbit around the Earth. Figure 3.39. Joe and Dan have completed the mission. Their program is shown below:

```
reset
set.boost.strength  250
set.time.step  40
go.until.time  7120
repeat  9  [boost  :rocket]
set.time.step  200
go.until.time  34500
repeat  7  [boost  :rocket]
go.until.time  214000
repeat  5  [boost.back  :rocket]
go.until.time  230720
repeat  2  [boost.back  :rocket]
go.until.time  250000
set.boost.strength  100
repeat  4  [boost  :rocket]
go.until.time  686720
repeat  1  [boost.back  :rocket]
go.until.time  1137520
repeat  5  [boost  :rocket]
go.until.time  1497520
repeat  10  [boost.back  :rocket]
start.animation
```

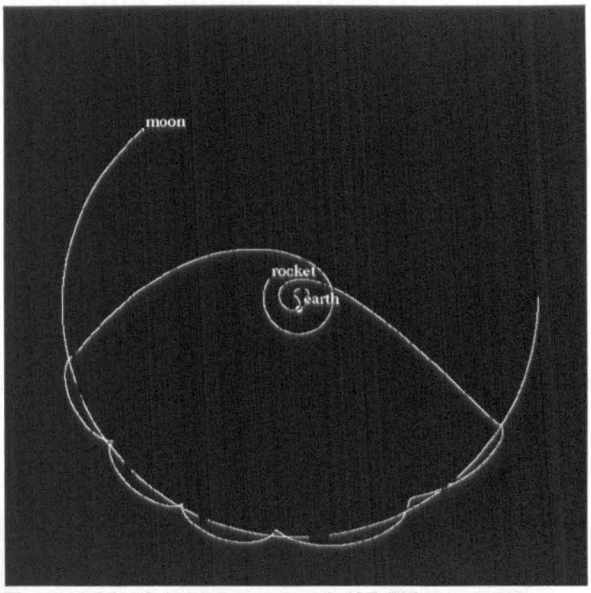


Figure 3.39 10 back boosts at 1,497,520 seconds.

3.3.3 *Discussion*

Joe and Dan have constructed a simple 22 line program which accomplishes a complex task: the launch and navigation of a rocket from the Earth to the Moon. They worked with Gravitas for less than five hours, including the time spent on the orbital procession phenomenon. Their solution is not particularly efficient - the rocket arrives at the Moon with excessive speed and they have to make more corrective boosts than is strictly necessary. However, the lunar transfer problem is an advanced topic in astrodynamics (Baker, 1967) and their solution works and could be improved at a later date - the program is a permanent record of their efforts.

The principal feature we wish to emphasise is the synergy between the two interfaces. There are eight boost sequences in their mission, and in each case Joe and Dan used the buttons of the graphical interface to try out different values of strength and timing. The programming interface allowed them to collect the boosts into a sequence which they could replay and edit at will.

Joe and Dan's program is, nevertheless, quite simple. It uses only seven of the fifty programming interface commands, and it does not use Logo in any complex way. Even at this level, we can see that programming is a useful tool, but in Appendix B we will see some of the more sophisticated purposes to which it may be put.

3.3.4 *Two Other Moon Trips*

To finish this chapter, we will summarise two other programs written to carry out the same mission.

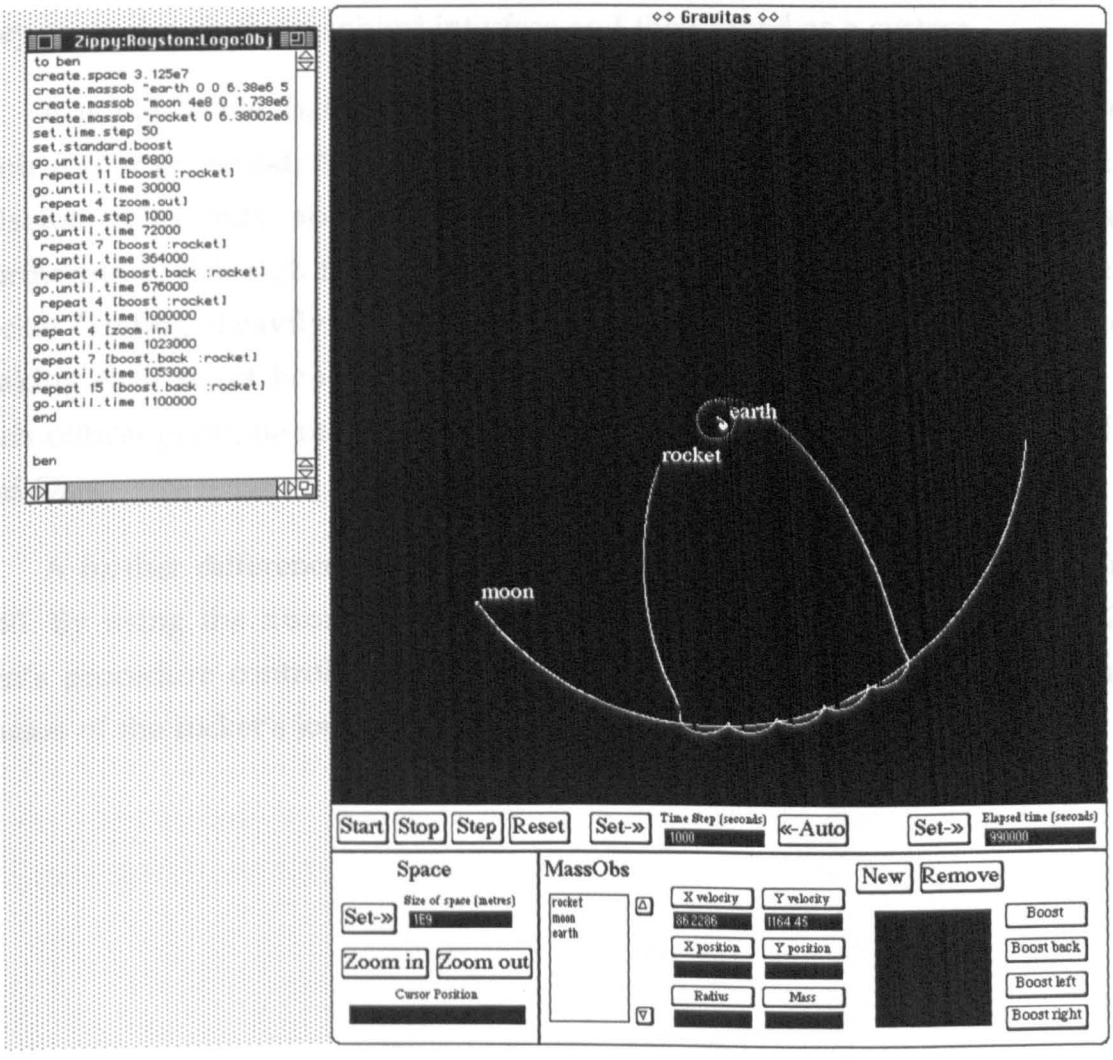


Figure 3.40 Ben's mission to the Moon and back.

Figure 3.40 shows the program (and the mission it generates) developed by a 14 year old school student called Ben. He took roughly the same time to produce it as Joe and Dan, about five hours, but had a slight advantage in that he had done some programming before (in Basic rather than Logo).

Ben's mission looks quite similar to Joe and Dan's, as can be seen by comparing figures 3.39 and 3.40. However, there are some important differences. First of all, his program is expressed as a Logo *procedure* called **ben**. The procedure is invoked by this name so that when the code in the editor at the left of figure 3.40 is run, **ben** is re-defined and then executed. This is a minor difference but it does mean that his procedure could be called by a higher level program which, for example, went on to take the rocket on a tour of more planetary objects.

Ben's procedure also creates the space and the three Massobs from scratch each time it is run. In contrast, Joe and Dan's program used Massobs which they created with the graphical interface and then saved as a system.

It is noticeable that Ben has the rocket orbit the Moon more closely, and with a shorter period, than Joe and Dan. He also managed to accomplish the mission with only six boost sequences, two fewer than Joe and Dan. Furthermore, although figure 3.40 does not show it, his mission ends with the rocket landing (heavily!) back on the Earth. In fact Ben was quite intent on making each set of boosts as accurate as he possibly could. Consequently, at each critical point, he made even greater use of the graphical interface than Joe and Dan.

A further difference cannot be seen in the snapshot represented by figure 3.40. By using the **zoom.out** and **zoom.in** commands at lines 11 and 20, Ben's procedure controls the size of the space so that the launch and return phases of the rocket's journey can be seen in detail.

In section 3.2.4 we gave the details of Simon's encounter with the orbital procession surprise. Here we describe his version of the voyage to the Moon.

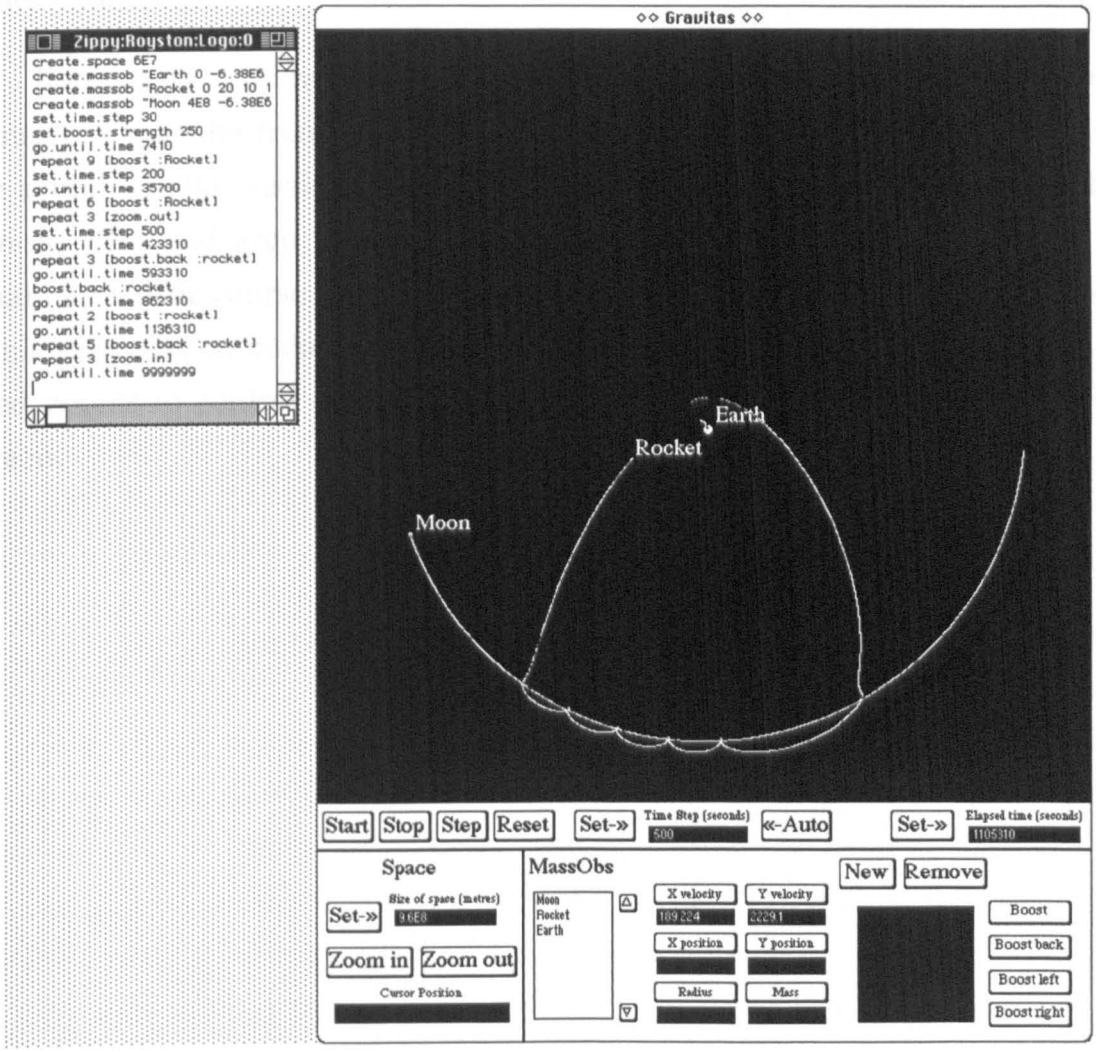


Figure 3.41 Simon's trip to the Moon.

There are a few points to note about Simon's mission. First of all, his journey is the most efficient of the three we have seen, in that it uses the lowest amount of boost. In practice this would mean his rocket would use less fuel. His boost to the Moon is only just strong enough, so the rocket requires a smaller deceleration boost for the Moon to capture it. Similarly, he then drops off the Moon and back to Earth with the smallest possible boost. In fact, if we remove the differences in launch velocity and final orbit from the missions we can compare the amount of boost used:

Simon	$6500ms^{-1}$
Ben	$6864ms^{-1}$
Joe and Dan	$7750ms^{-1}$

We should stress that efficiency in the use of boosts was not an element of the task set by the researcher, but was a self imposed aim for Simon, and to a lesser extent Ben.

Returning to Simon's mission, a result of his minimum boost lunar transfer is that the transfer trajectory is more curved and therefore more like the actual Apollo missions. Again, this was not a requirement of the task, but Simon, who had seen diagrams of the Apollo Moon shots, was pleased to obtain a realistic course. He also made a corrective boost to the rocket, at line 17 of his program, to make its orbit around the Moon less eccentric. Like Ben, he used zooms to control the size of the space so that the Earth orbits filled the screen.

3.4 *Summary*

In many ways, the studies we have described raise more questions than they answer. What sort of physics knowledge are the subjects applying? What do they gain from Gravitas? What could they accomplish without the assistance of an expert? However, this was always the intention. Massobs are new entities and we wanted to carry out a serious examination of their scope for the exploration of some physical concepts. The studies put us in a position to pose questions such as those above, and we have found contexts in which to research them.

We have shown that using Gravitas is primarily a *constructive* activity. Learners can build systems of Massobs and programs which control them, and see what they do. We have seen that users are frequently surprised by the things they have built, and these surprises may be gateways to profound physical insights.

The Lunar Journey demonstrates a useful synergy between the two interfaces. It indicates that tasks which are beyond the scope of a single mode interface may be rendered feasible. Gravitas is not the only kind of system where such an enhanced interaction is fostered. HyperCard for the Apple Macintosh is a system which may be controlled with its graphical interface and by programs written in its embedded language, HyperTalk. The point about Gravitas is that the graphical interface makes Massobs easier for learners to comprehend, since they can be picked up and boosted by simple mouse actions, while their interface to a standard and popular programming language, Logo, opens a realm of educational possibilities.

4 Gravitas and the School Curriculum

4.1 Overview

The previous chapter showed Gravitas being used for tasks which, although educationally meaningful, were suited to the specific purposes of our investigation. In this chapter we wish to examine the possibilities for uses of Gravitas in more realistic settings. A reasonable question to ask is "How could Gravitas be used to illustrate topics in school science courses"? In the United Kingdom the government has set down a National Curriculum for Science (Department of Education and Science, 1991), and although this curriculum is currently under review we will employ it as a guide to illustrate how Gravitas might be used. We will then move on to show that Gravitas' programmability opens a door onto a range of more open-ended projects for the science classroom. Finally, we will emphasise that programmability also makes Gravitas *extensible* in that its in-built functionality can be augmented with procedures written in Logo.

4.2 *Gravitas and the National Curriculum for Science*

In this section we will we survey the National Curriculum for Science (Department of Education and Science, 1991) and point out the areas where Gravitas based activities seem particularly natural. In its 1991 form the curriculum is broken into four *Attainment Targets* – AT1: Scientific Investigation; AT2: Life and Living Processes; AT3 Materials and their Properties; AT4: Physical Processes.

AT4 is the component for which Gravitas has the most relevance and is therefore the section we will concentrate on. Within the Attainment Target there are 10 *Levels*, each of which contains several *Statements of Attainment* and some corresponding suggested activities. Together with a brief discussion of the aims and general nature of the programme of study, these Statements of Attainment make up the curriculum content. There is an additional layer of complexity added by the structuring of the curriculum into four *Key Stages* which define the level within the Attainment Targets that average children of certain ages should be expected to reach. However, this division need not concern us in this survey since it does not affect the overall curriculum content. Our suggestions are pitched at children of age 13 or 14, the ages of most of our subjects in the chapter 3 studies, although practical testing is necessary to check their feasibility. We begin at Level 2, the lowest level at which we believe Gravitas could be of use.

4.2.1 *Level 2*

Statement C

"Pupils should understand that pushes and pulls can make things start moving, speed up, slow down or stop."

If we explain Gravitas' boost commands as 'pushes' and 'pulls' then there are many ways to demonstrate this statement, both interactively and programmatically. For instance, simply by holding down the boost button a user will see the selected body accelerate. Figure 4.1 was produced by doing this until Massob A reached the centre of the screen and then holding down instead the boost back button. The spacing of the dots clearly shows the increase and decrease in speed, which is almost symmetrical because boost and boost back are of the same magnitude.

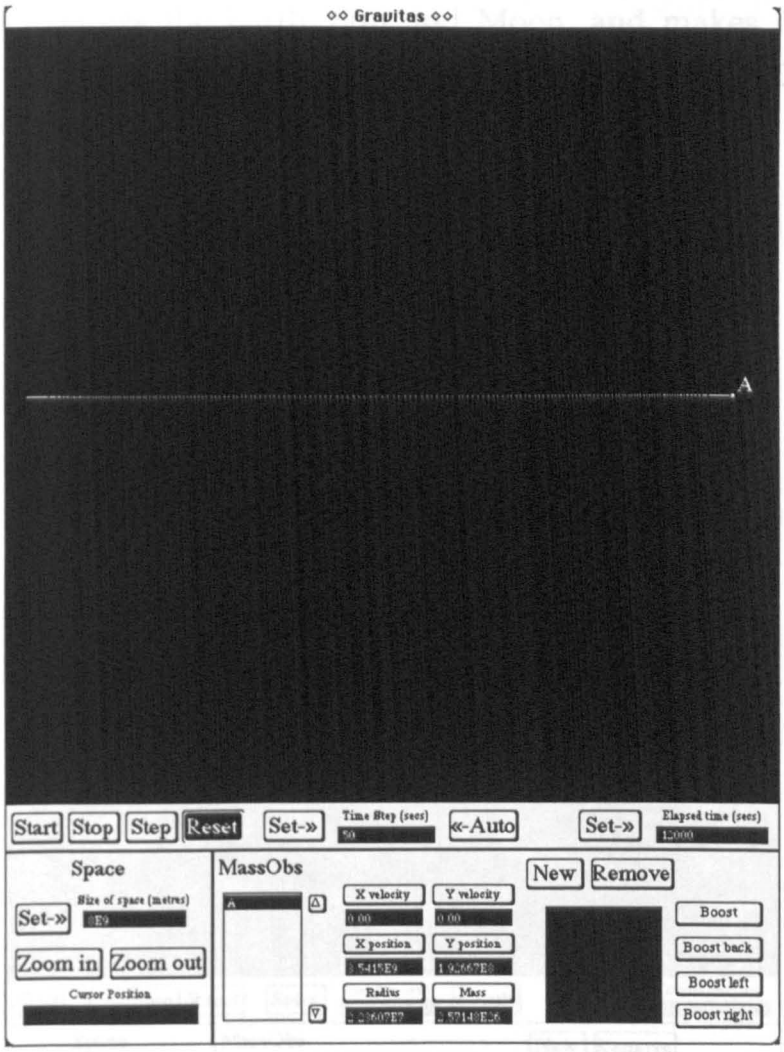


Figure 4.1 Accelerating and decelerating a massob with boosts

The same image could be produced by two lines of Logo:

```
repeat 120 [boost :A step.animation]
repeat 120 [boost.back :A step.animation]
```

Statement E

"Pupils should know that the Earth, Sun and Moon are separate spherical bodies."

This may be well illustrated by Gravitas, but of course, only in two dimensions in the current version. In fact, although for technical reasons the images in this thesis show massobs as flat white discs, on colour screens they are rendered (when large enough for it to make a difference) to look like spheres.

Figure 4.2 shows the Earth, Sun and Moon, and makes the important point that if we zoom out far enough to see the whole system then the Earth and Moon are almost impossible to separate.

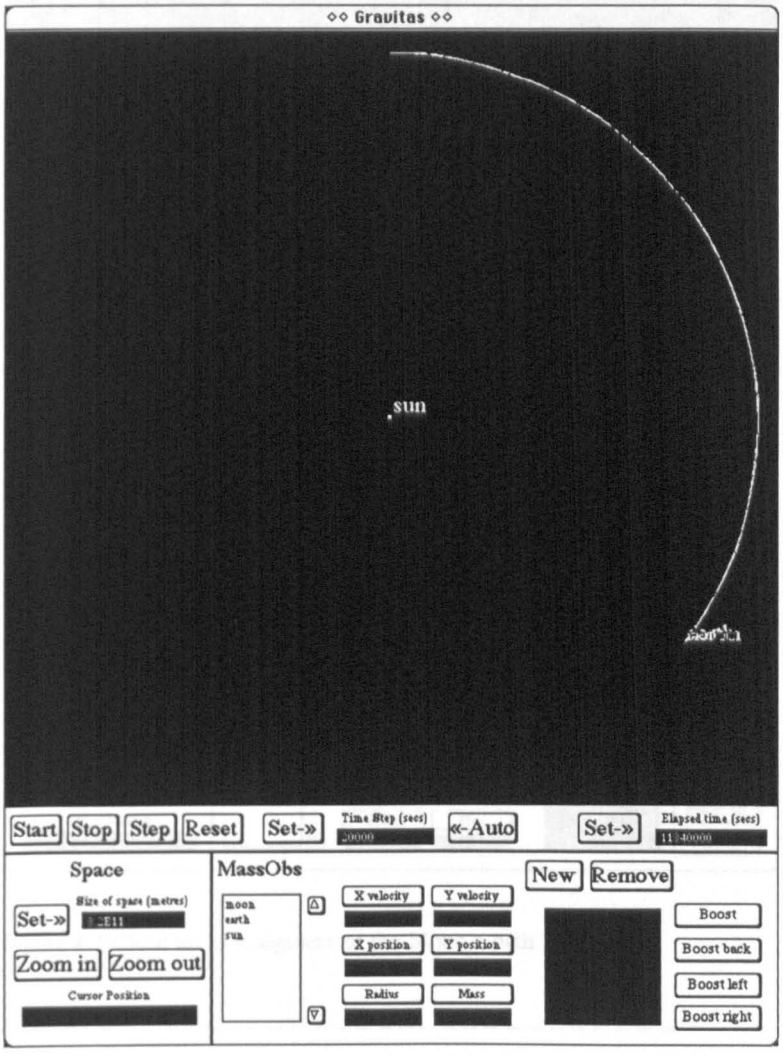


Figure 4.2 Earth and Moon orbiting the Sun

This leads to the question of the moon's path, which we cannot make out at this range. However, it is a simple matter to zoom and pan (there is a **pan** command in the Programming Interface) Gravitas until we can examine a segment of the path in greater detail. Figure 4.3 overleaf shows an example.

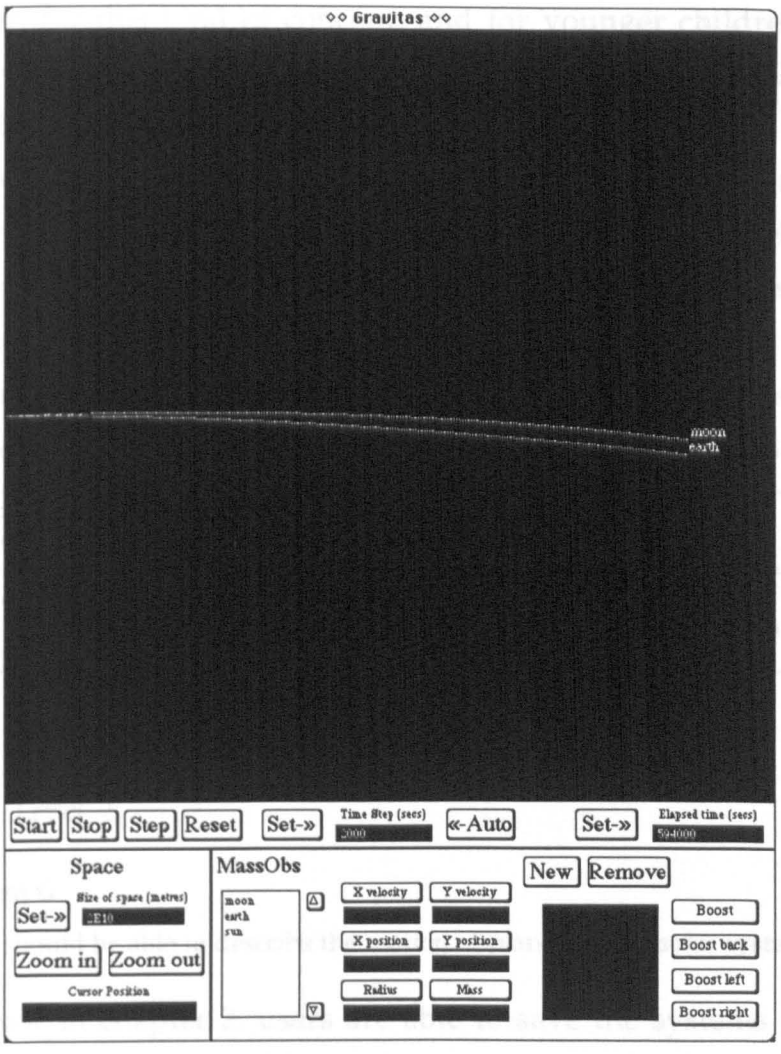


Figure 4.3 Close up of a segment of the Moon's path

4.2.2 *Level 4*

Statement C

"Pupils should know that more than one force can act on an object and that forces can act in different directions."

This concept can be demonstrated in many ways with the current version Gravitas. Firstly, it is a natural occurrence once two or more massobs have been created – the gravitational forces and any boosts have a combined effect. Secondly, if, as was found to be a good idea in the studies of chapter three, the user is introduced to the idea of 'stacking' boosts while the system is paused then several forces can be applied at the same instant. A plain boost and a boost right add up to a 'diagonal' boost.

However, for this kind of concept, and for younger children, it could be more effective to add a *directed* boost command which takes an extra parameter: the heading along which the boost is to act. This would bring Gravitas, which is tuned at present for investigations in orbital mechanics, into line with systems such as diSessa's Dynaturtles (diSessa, 1982) and Brna's ROCKET (Brna, 1989). Since the programming interface has a complete window onto the state of any Massob, such commands would be straightforward to implement. An example is given below for a scheme where the boost heading is given in degrees, zero degrees pointing up the screen.

```
to aim.boost :massob :heading
set.xvel :massob      (xvel :massob)
                  +   (boost.strength * sin :heading)
set.yvel :massob      (yvel :massob)
                  +   (boost.strength * cos :heading)
end
```

4.2.3 *Level 5*

Statement G

"Pupils should be able to describe the motion of planets in the solar system."

As we saw in chapter 3, users are able to save the systems they create in Gravitas. Since building Gravitas we have created and saved many example systems in which the Massobs represent real astronomical objects. One of these is a model of the solar system. Students can load this system and examine it in detail. They can zoom in and out, move around the system, make measurement of distances and orbital periods, all with a few mouse clicks. In fact, we have constructed several solar system models in Gravitas, two of which are particularly interesting. The first is a snapshot of our solar system as it stood in March 1991 (constructed from ephemeris tables) when an early Gravitas prototype was finished. This is an interesting view as it shows Pluto closer to the Sun than Neptune, an infrequent occurrence. The second version shows all the planets in a line at their average distances from the sun. Such a configuration never occurs in nature but it does permit some interesting comparisons of relative distances and orbital velocities. Figure 4.4 shows an example of this system. The inner (or *Terran*) planets have been told not to show their labels and Mars is just about to complete an orbit. This represents about 654 days of simulation.

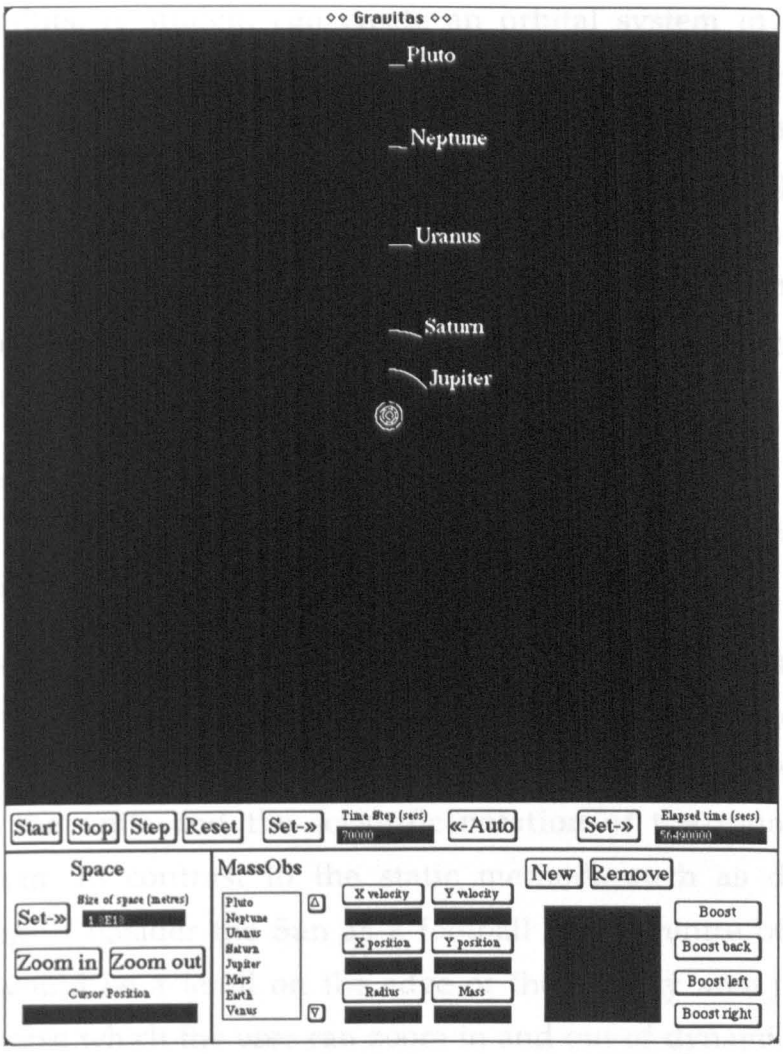


Figure 4.4 The solar system with all planets at their average distance from the sun.

4.2.4 *Level 6*

Statement B

"Pupils should understand that energy is conserved."

The law of Conservation of energy is of course fundamental to physics. It also has huge scope and there are many ways in which it may be applied within the framework of Gravitas. However, some especially interesting lines of investigation are opened up by two of Gravitas' *tool* procedures – `potential.energy` and `kinetic.energy`. These two procedures, which are described in more detail later in this chapter, take a list of Massobs as their input and produce the total gravitational potential energy and the kinetic energy of the list respectively. Of course, these concepts move this suggestion beyond the understanding most 13 or 14 year olds and into the last two years of GCSE. Nevertheless, an important manifestation of the conservation law can

be seen in orbits. A student can create an orbital system in Gravitas and examine the kinetic energy of the bodies by typing `print kinetic.energy [A B]` at intervals as the orbit progresses. But this quantity will be seen to vary in an eccentric orbit – energy is apparently *not* being conserved. The explanation is that it is the *total* energy in an orbit which is conserved – the sum of potential and kinetic energy. Typing the command `print kinetic.energy [A B] + potential.energy [A B]` at intervals around the orbit will verify this.

Statement G

"Pupils should know that the solar system forms part of a galaxy which is part of a larger system called the Universe."

Gravitas can offer a new approach to the task of introducing students to the huge scale of the Universe. The distances involved range over many orders of magnitude and can be difficult to appreciate even when a child understands the units and the scientific notation of the numbers used to represent them. In contrast to the static methods such as diagrams and metaphors (eg. Consider the Sun as a football in the centre of a pitch. The Earth then, would be a lentil on the edge of the penalty area.) Gravitas can depict a Universe which the user can zoom in and out of dynamically.

Of course, not all the objects in the Universe can be stored, at present Gravitas can handle only a few hundred Massobs. Nevertheless, we have created one example which contains the solar system, then a few nearby stars, then a hundred or so stars to represent our galaxy and finally several galaxies (each represented by a few Massobs). The user can begin with the Sun almost filling the screen and then zoom smoothly out to watch the planets come into view. By the time Pluto is on the screen the inner planets seem almost on top of each other and the Sun is just a dot. A few more zooms (each one doubles the size of the space) and the solar system has receded to a single dot just as the nearest star comes into view. It takes another 15 zooms to get the whole of our galaxy onto the screen, and we are now looking at a region about 100,000 light years across. Five more zooms bring our nearest neighbour galaxies into sight.

In this mode, Gravitas is simply being used as an animated star map and not even an accurate one as the galactic stars are illustrative rather than real. However, we believe the journey described above is another way to communicate astronomical distances to learners.

4.2.5 *Level 7*

Statement G

"Pupils should know that gravity acts between all masses and the magnitude of the force diminishes with distance."

Gravitas is well suited to the illustration of this concept. All the Massobs created by users interact gravitationally and it is possible to set up situations which show the truth of the second part of Statement G. Figure 4.5 shows an example:

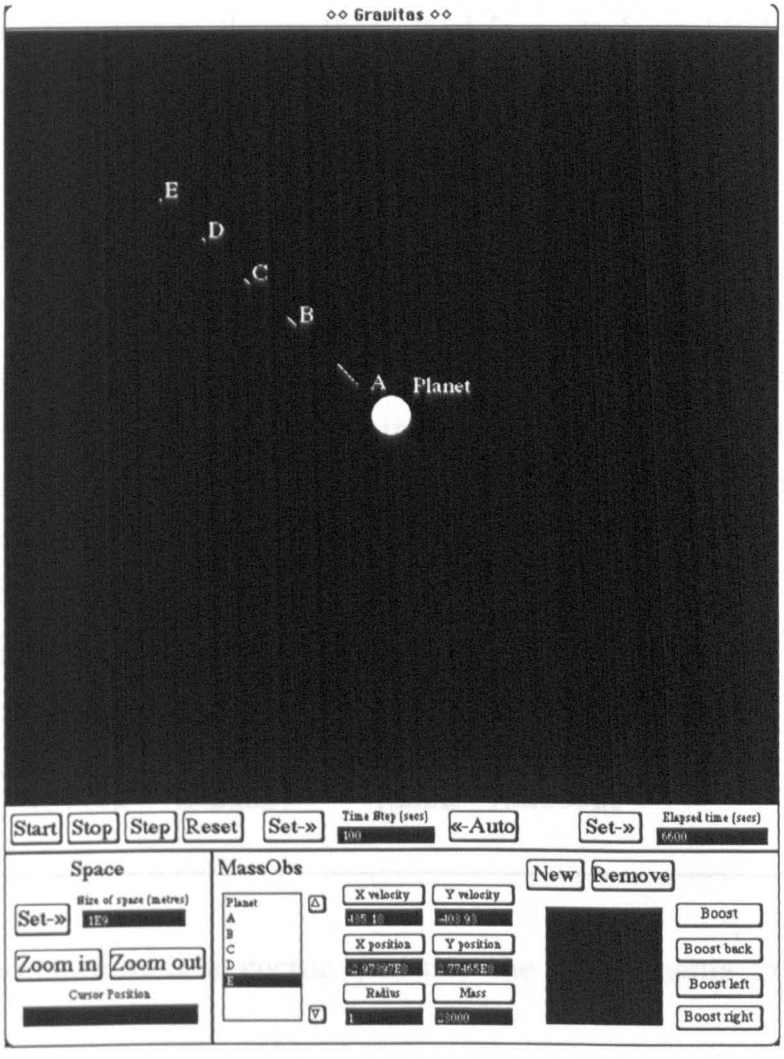


Figure 4.5 An illustration of gravity diminishing with distance

The Massobs A, B, C, D and E are all light enough that we can ignore their gravitational effect on each other. Their trails show that acceleration is greater the nearer the Massobs are to the planet.

4.2.6 *Level 9*

Statement E

"Pupils should be able to relate the theory of gravitational force to the motion of satellites ."

One important concept in this area is the idea of geo-synchronous satellite orbits, that is, satellites which orbit the Earth in exactly the time it takes the Earth to rotate on its axis. With Gravitas, students can be shown that the Moon takes 28 days to go round us, but that a low orbit satellite can take just a few hours. Somewhere in between then, there must be a position which has an orbital period of 24 hours. This point can be found by trial and error with Gravitas as a precursor to the mathematical formula for orbital period which they will be taught.

4.2.7 *Level 10*

Statement C

"Pupils should understand the concept of momentum and its conservation."

We showed one novel manifestation of momentum conservation in chapter 3 – the orbital procession surprise. It is also illustrative to create systems of stationary Massobs and let the accelerate towards each other. With the help of procedures such as those below, students can verify that the total momentum of such systems is always zero.

```
to x.momentum :massob
output (x.vel :massob) * (mass :massob)
end

to y.momentum :massob
output (y.vel :massob) * (mass :massob)
end
```

Since momentum is a vector quantity the components must be added separately.

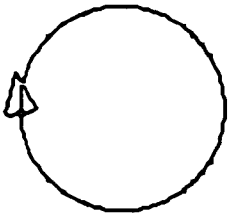
4.3 *Sample programs*

This section explores a few more of the possibilities opened up by Gravitas' programmability. The ideas discussed here do not map neatly onto statements or targets from the National Curriculum but could be used as discussion material for many of its concepts or as starting points for longer term projects. Our intention is to show that programmability gives Gravitas the potential for greater educational utility. In chapter 3 we showed user programs which navigated a rocket between the Earth and the Moon. In this section we wish to hint at wider possibilities.

4.3.1 *The Massob Spiral*

The following procedure was written by a user who was shown Gravitas at an early stage of its development. This user was quite familiar with Turtle Geometry and Logo and he began by thinking about the well known procedure for a circle (see, for example Papert, 1980 p58):

```
to circle
forward 1
right 1
circle
end
```



When this procedure is run the turtle draws a circle on the screen and moves around it forever. The user tried building an analogous procedure in Gravitas, thinking naively that perpetually boosting a Massob on one side would make it draw a circle too:

```
to circle
step.animation
boost.right :fred
circle
end
```

In fact, the Massob generated the pattern shown in figure 4.6, a spiral.

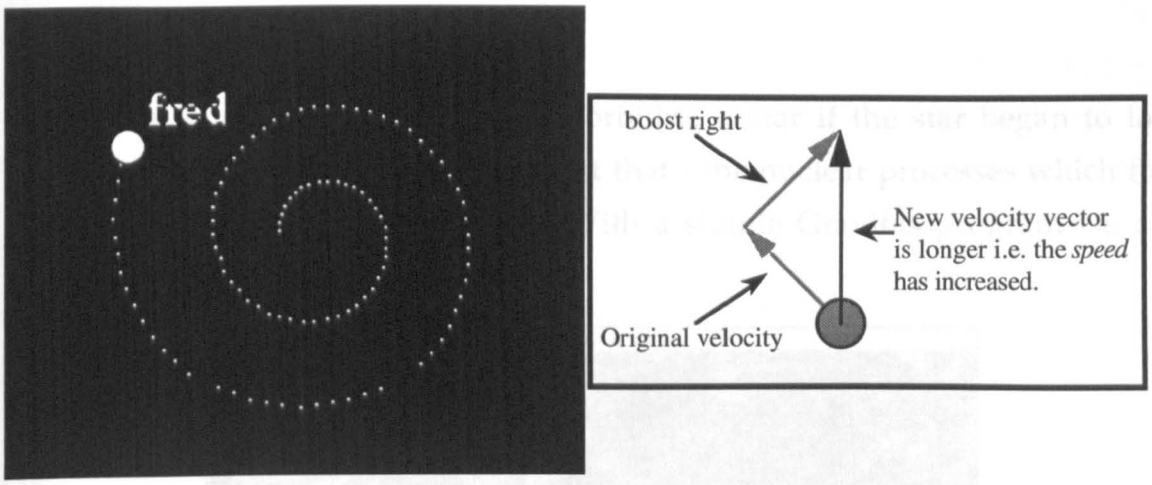


Figure 4.6 The effect of continuous boost right

The user was rather surprised by this at first, until, in conjunction with the system's designer, he considered exactly how **`boost.right`** works.

The command **`boost.right`** acts at right angles to a Massob's direction of travel and has no effect on the velocity in that direction. But in adding a new, orthogonal component to the old velocity, the command inevitably increases the Massob's *speed*. The **`boost.strength`** however, is constant, so a subsequent boost will rotate the velocity vector through a smaller angle and increase the speed by a smaller amount. The net effect is that the Massob traces out a spiral. The procedure immediately raises several questions: What is the gap between successive orbits? Is the spiral equiangular or logarithmic? Will it go on for ever? Is it *possible* to make a Massob travel in a circle with a simple program?

4.3.2 *The Evaporating Planet*

What would happen to a planet orbiting a star if the star began to lose mass? In fact, our own Sun is doing just that - the nuclear processes which fuel it result in a continual loss of mass. With a simple Gravitas program we can investigate this situation:

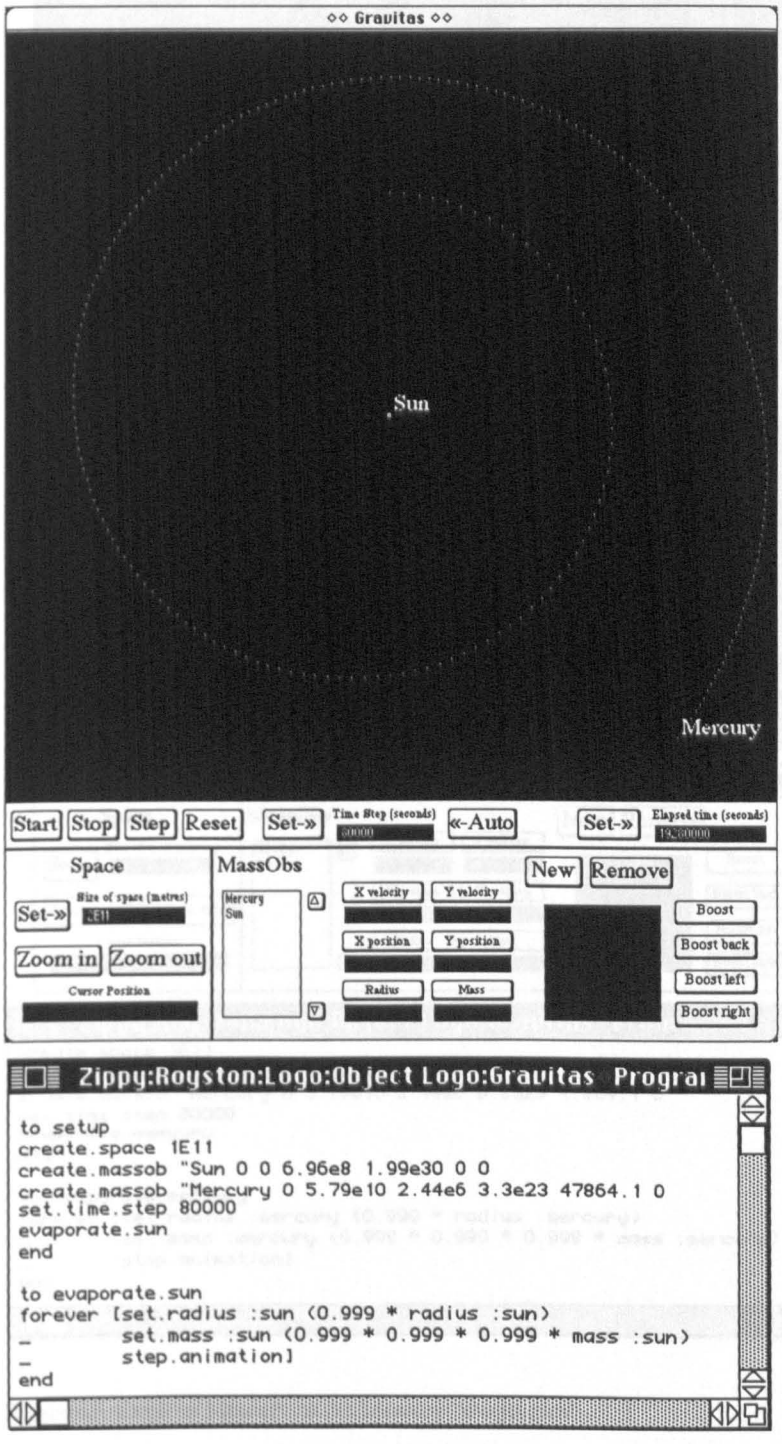


Figure 4.7 Orbit of Mercury as the Sun "evaporates".

As we can see, Mercury spirals away from the Sun. Of course, the situation is highly exaggerated. The Sun is actually losing mass at a rate that is

tiny compared with that in the program. The way that **evaporate.sun** is set up, the Sun loses one tenth of a percent of its mass every 80,000 seconds. This is about ten orders of magnitude faster than is the case! Nevertheless, the procedure is interesting, and it does prompt a complementary question: What happens if it is the planet which is evaporating?

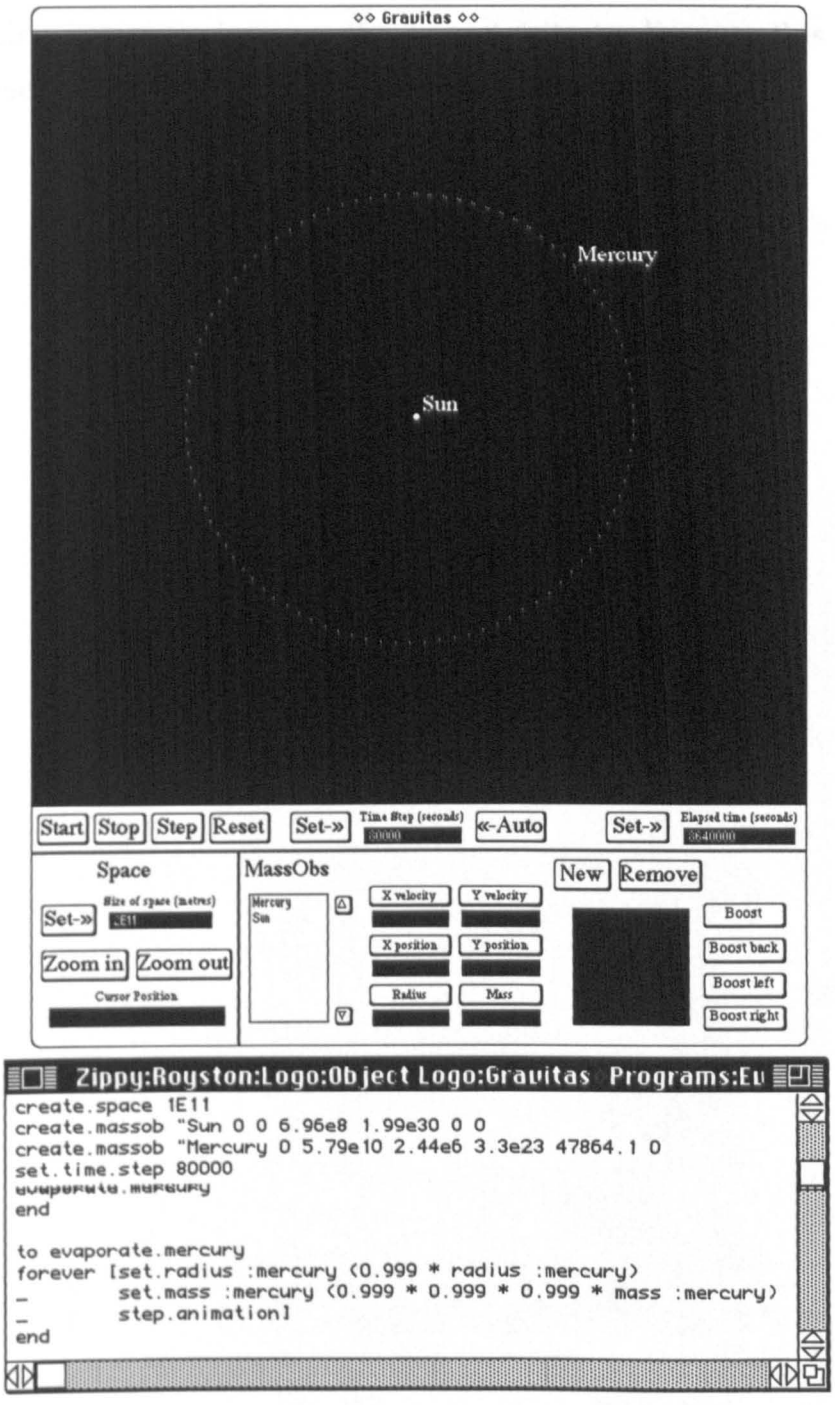


Figure 4.8 Orbit of Mercury as the Mercury "evaporates".

The answer is that very little happens. Mercury continues in the same circular orbit. This sequence leads neatly into a key astronomical insight: if we can measure the orbit of a satellite - its radius and period - then we can

calculate the mass of the object it is orbiting. In other words, the orbital components of a satellite are independent of its mass, *provided* that its mass is small with respect to the object it is orbiting.

This result is a mathematical consequence of Newton's Theory of Gravitation, which astronomers have long known about. However, we have shown that Gravitas gives learners the opportunity to discover this important fact for themselves.

4.3.3 *A Star Cluster*

Imagine we wished to examine the dynamics of a cluster of stars. It would be tedious to create fifty or a hundred Massobs by hand. A straightforward procedure can accomplish the task without fuss:

```
Zippy:Royston:Logo:Object Logo:Gravitas.Work
to random.stars :names
if emptyp :names [stop]
localmake "radius (random 10) * 1e8
create.massob first :names
_             (random 1000) * 1.5e8 * (1 - 2 * random 2)  ;;x-position
_             (random 1000) * 1.5e8 * (1 - 2 * random 2)  ;;y-position
_             :radius                                     ;;radius
_             1e4 * :radius * :radius * :radius           ;;mass
_             (random 10) * 1e4 * (1 - 2 * random 2)      ;;x-velocity
_             (random 10) * 1e4 * (1 - 2 * random 2)      ;;y-velocity
random.stars butfirst :names
end

to star.cluster
create.space 2e11
random.stars [s1  s2  s3  s4  s5  s6  s7  s8  s9  s10
_             s11 s12 s13 s14 s15 s16 s17 s18 s19 s20
_             s21 s22 s23 s24 s25 s26 s27 s28 s29 s30
_             s31 s32 s33 s34 s35 s36 s37 s38 s39 s40
_             s41 s42 s43 s44 s45 s46 s47 s48 s49 s50]
end
```

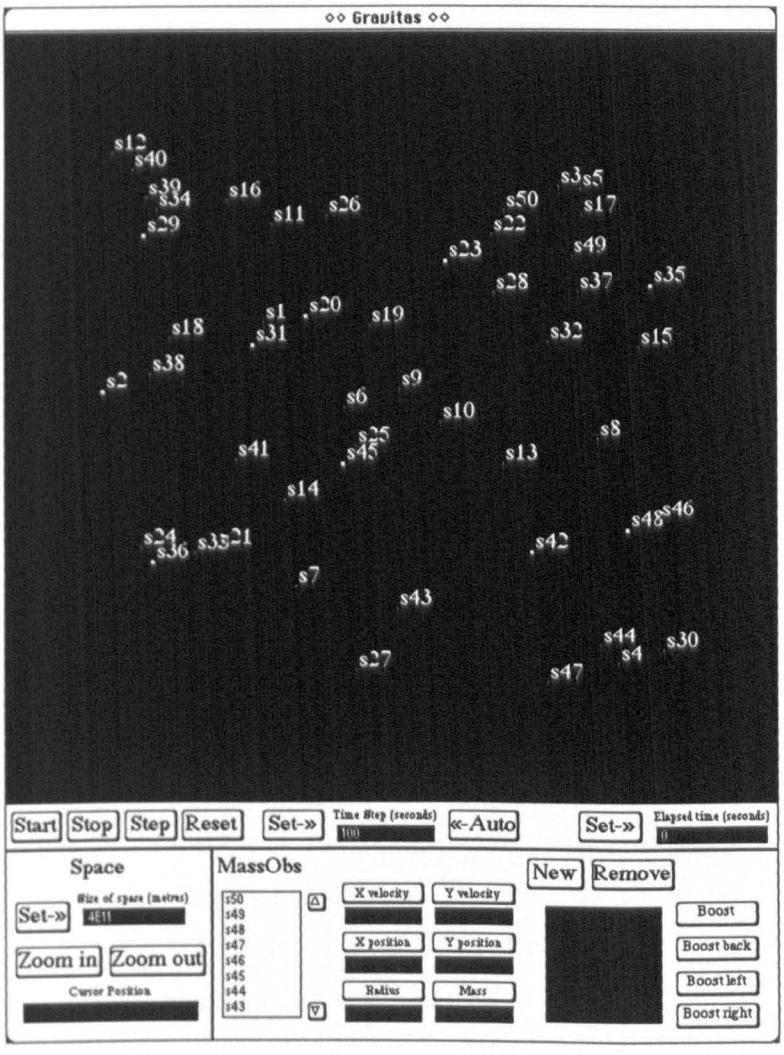


Figure 4.9 A star cluster.

The procedure, **star.cluster**, calls a "manufacturing" procedure, **random.stars**, which takes a list of names as its input. **random.stars** then creates a Massob for each name and gives it a random position, size, mass and velocity. These "stars" are up to five times as massive as our own Sun.

A cluster such as this could be used to illustrate quite sophisticated astronomical concepts, such as relaxation time and the virial theorem (Roy, 1978).

4.3.4 *Collision Detection and Planet Formation*

When two Massobs collide they coalesce to form a single new Massob which travels on with the combined momentum. This behaviour can be altered because the collision handler is a *public* procedure which users are free to edit or replace. In this way, Massobs could be made to bounce off each other, or fragment. The default behaviour though, can lead to interesting situations.

For example, two of the pioneers of dynamical astronomy, Myron Lecar and Sverre Aarseth, decided to examine a theory of the formation of planets around a star (Lecar and Aarseth, 1986). They used a computer model similar to the mechanism which gives Massobs their behaviour to predict what would happen to 200 Moon sized objects orbiting the Sun in a belt stretching from 0.5 to 1.5 Astronomical Units (1 A.U. is equal to the radius of the Earth's orbit). They found that after a period of 50,000 years, coalescent collisions between planetesimals resulted in the formation of six planet sized bodies. We decided to apply Gravitas to this same experiment. First of all we built procedures to create the Sun and the planetesimals:

```
to setup
create.space  3e11
create.massob  "Sun
_              0            ;;x position
_              0            ;;y position
_               6.96e8      ;;radius
_              2e30         ;;mass
_              0            ;;x velocity
_              0            ;;y velocity
;;
generate.planetesimals  200
;;
end
```

This procedure creates a Massob to represent the Sun and places it stationary at the centre of coordinates. It then calls a sub procedure - **`generate.planetesimals`** - to create 200 planetesimals, randomly positioned within the belt mentioned above.

```
to generate.planetesimals :num
if equalp :num 0 [stop]
generate.one :num
generate.planetesimals :num - 1
end
```

generate.planetesimals is a tail recursive procedure which calls **generate.one** the number of times specified in its input and then stops.

```
to generate.one :num
localmake "theta random 360
localmake "r 7.5e10 + (1.5e9 * random 100)
localmake "name word "p :num
create.massob :name
_              :r * cos :theta
_              :r * sin :theta
_              2e6
_              2e23
_              0
_              0
standard.orbit :sun massob :name
end

setup
```

generate.one calculates a random position in the belt, in polar coordinates (which are easier to understand in this context). It then "invents" a name for the new planetesimal by appending the index to the letter "p". The call to **create.massob** actually creates the new Massob, converting the polar coordinates to Cartesian and setting the radius and mass to those of the Moon. Finally, the new Massob is given the velocity for a standard (i.e. circular) orbit around the Sun by a call to the **standard.orbit** tool procedure (described in the next section). Running **setup** produces a space like figure 4.10.

Unfortunately, Gravitas cannot run this configuration of Massobs quickly enough to actually duplicate Lecar and Aarseth's experiment. The best it can do on a Macintosh IIfx, while retaining reasonable accuracy, is accelerate real time by a factor of around 50,000. So Lecar and Aarseth's simulation would take about a year of full time running! However, all is not lost. First of all, just running it for a few days produces perhaps 3 or 4 collisions, from which an overall result could be extrapolated. Second, Lecar and Aarseth give details of the optimisation techniques they used to get adequate performance on their computer (which was about 10 times as fast as ours) and these could, with some work, be incorporated into Gravitas. And of course, faster computers are appearing all the time!

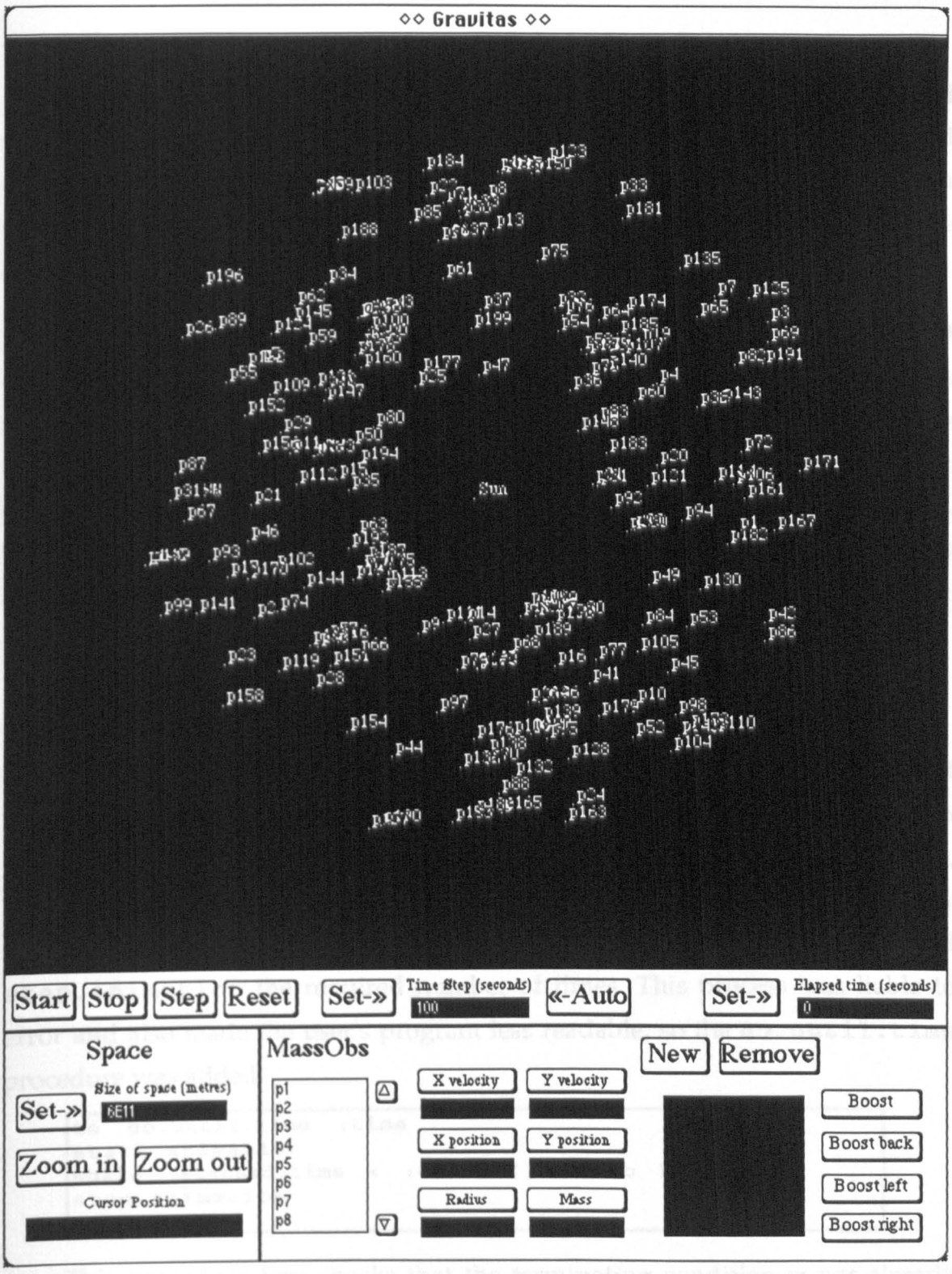


Figure 4.10 200 planetesimals orbiting the Sun.

4.4 *Tools*

During the development of Gravitas we have built many *tools*, small procedures which carry out some useful task. There is no need for a great deal of specialist knowledge to build tools, as the medium for the extensions is a standard and common language - Logo. We will illustrate this with some examples.

During one videotaped session a user became interested in the *speed* of one of the Massobs he had created. But speed is not part of a Massob's representation, only the x and y velocity components. Of course, a simple mathematical relationship exists between speed and the velocity components: speed is equal to the square root of the sum of the squares of the components. The programming interface contains commands that access a Massob's velocity components and so it was a simple matter to construct a 'speed meter' in Logo:

```
to speed :massob
output sqrt ((xvel :massob) * (xvel :massob) +
            (yvel :massob) * (yvel :massob))
end
```

Another extension was built onto Gravitas to make a common feature of user's programs less clumsy. The interface procedures **start.animation** and **stop.animation** can be used to turn the system on and off from Logo, but if the user wanted the animation to run *until* a given time it was necessary to calculate how many time steps it would take and then call **step.animation** the required number of times. This process was liable to error and also made the user's program less readable, so the **go.until.time** procedure was added:

```
to go.until.time :time
start.animation
while (elapsed.time < :time) [] ;; Do Nothing
stop.animation
end
```

This procedure first checks that the terminating condition is not already met and then starts the animation. The procedure then does nothing until the condition is satisfied when it wakes up and stops Gravitas.

Two concepts which are important to the physics of gravitating systems are the kinetic and potential energy of the ensemble. We have constructed tools which calculate each of these quantities for any list of Massobs:

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;Returns the total kinetic energy in a group ;;
;;of Massobs                                 ;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
to kinetic.energy :massobs
if emptyp :massobs [op 0]
if objectp :massobs [op .5 * (mass :massobs)
                          * ((xvel :massobs) ^ 2 +
                            (yvel :massobs) ^ 2)]
op .5 * (mass  first :massobs)
      * ((xvel first :massobs) ^ 2 +
         (yvel first :massobs) ^ 2)
      + kinetic.energy bf :massobs
end
```

This recursive procedure works with either a single Massob or a list as its input. Thus **`kinetic.energy massobs`** would return the total kinetic energy of the current system. Another tool procedure, **`potential.energy`**, works in the same way.

In the previous section we saw the use of a standard orbit calculator. Given two Massobs as input, it sets the velocity of the second to a value which gives it a circular orbit around the first.

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;Gives m2 the velocity for a circular       ;;
;;orbit around m1                            ;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
to standard.orbit :m1 :m2
localmake "r separation :m1 :m2
localmake "rx (xpos :m2) - (xpos :m1)
localmake "ry (ypos :m1) - (ypos :m2)
set.xvel :m2 (xvel :m1) +
_               (sqrt ((big.g * mass :m1) / :r)) *
_              :ry / :r
set.yvel :m2 (yvel :m1) +
_               (sqrt ((big.g * mass :m1) / :r)) *
_              :rx / :r
end
```

However, a heavy planet will not orbit a light one, so this procedure gives strange results if the second Massob is not much less massive than the first.

Something most of our subjects asked during the "moon shots" of chapter 3, was "Do the boosts use up fuel?" In Gravitas as it stands, they don't, but it is not difficult to construct new ones which do:

```
to refuel
make "fuel.load 100
end

to new.boost :massob
if :fuel.load < 1 [print [Out of Fuel!!] stop]
boost :massob
make "fuel.load :fuel.load - 1
end
```

The procedure **new.boost** will only allow 100 boosts before refuelling is necessary. However, this constraint applies globally, so any boost of any Massob depletes the fuel load. Fortunately, the dialect of Logo attached to Gravitas allows variables to be personally allocated to Massobs. This means we can give individual Massobs their own fuel:

```
to refuel :massob
ask :massob [havemake "fuel.load 100]
end

to new.boost :massob
if (ask :massob [:fuel.load]) < 1
_   [print list (name :massob) [is out of Fuel!!]
_    stop]
boost :massob
ask :massob [make "fuel.load :fuel.load - 1]
end
```

The **ask** and **havemake** constructs allow us to define private variables for each Massob. Of course, these can be used in any context, not just fuel using boosts and they provide a very powerful *general* means of extending Gravitas' capabilities.

4.5 *Summary*

In this chapter we have shown that there are a number of areas in the Science National Curriculum for which it is possible to devise Gravitas based activities. The Statements of Attainment for which Gravitas seems particularly well suited are those concerned with concepts such as Force, Momentum, Energy and Gravity, and with more general knowledge of astronomical bodies and satellites.

We have also indicated that the programmable nature of Gravitas opens up wider possibilities. First, it makes it possible to build Logo procedures which, although simple, can be the vehicle for longer term investigations. Although this possibility was also described in chapter 3, in this chapter we have shown that it applies to more than just orbital transfers.

Secondly, we have described the way in which Gravitas may have its basic functionality extended through the addition of Logo procedures which take advantage of the programming interface's complete window onto the state of Massobs and the space.

The University of Sheffield

Department of Psychology

Dr J P Frisby (Head of Department)
Prof K J Connolly
Prof J EW Mayhew
Prof P K Smith

PO Box 603 Psychology Bldg
Western Bank Sheffield
S10 2UR UK
Phone 0742 826558
Fax +44 (0) 742 766515
Telex 547216 UGSHEFG
E-mail PC1AC@UK.AC.Shef.PA

17th August 1994

Dear Mrs Robinson,

I have finally had chance to get some information copied for you regarding early signs of dyslexia. The information which I enclose comes from the literature I have built up for the "Eraly Identification" project, but I am afraid it may not be too useful to you as there is *very little* clear information on early diagnosis, as nobody has really found a way of going about it yet.

So, I have included 3 articles written by Jean Augur, the first is taken from a copy of "Dyslexia Contact" from June 1990, the second is from a book called "Children's Written Language Difficulties" edited by M.J.Snowling (1985). The third piece is a leaflet from the B.D.A. Jean Augur was the Education Officer for the B.D.A., but she sadly died a short while ago. She seems to have had a very good understanding of the characteristics of dyslexia - she actually had 3 dyslexic sons herself. A talk that she gave at the beginning of 1991 was the major inspiration for my early identification project.

The other pieces of information come from research and other more academic aspects of dyslexia. The first is a chapter from a book called "Children's Reading Problems" written by Peter Bryant and Lynette Bradley. It has information regarding the development of reading skills. The other section is 3 chapters from a book by Peter Pumfrey and Rea Reason called "Specific Learning Difficulties (Dyslexia): Challenges and Responses". It is the result of a national inquiry about dyslexia with researchers and educational psychologists.

I hope some of this information is of use to you. Please let me know if you want me to look out for more. My new address is :

The Department of Psychology
University of Manchester
Oxford Road
Manchester
M13 9PL

I hope things go well for you. I will send you a copy of the findings of the study when I have them. I would be really grateful if you could send me a copy of the diagnostic reports for Thomas.

Regards.

Sue Pickering.

5 Objectworlds and other Educational Computing Systems

5.1 Overview

In chapter one we defined a particular class of learning environments, and named them objectworlds. In the second chapter we described a new member of the class, called Gravitas, and in the third and fourth we showed what kind of educational activities it can support. In this chapter we wish to set objectworlds in context by contrasting them with other kinds of systems. One underlying reason for doing this is to provide some help for teachers who must choose software to use with their students.

In the interest of brevity, it makes sense to compare objectworlds with their close relatives among the family of educational computing systems. To this end we will begin the chapter with a short classification, concentrating on programs which, like Gravitas, offer their users varieties of *discovery* learning in scientific domains.

The two main categories we identify as close to objectworlds are called Modelling Systems and Simulations, and, after clarifying these terms, we will use most of the rest of the chapter to describe some well known examples, emphasising the differences in the kinds of activities each can foster. We will round the chapter off with a synthesis of the ideas covered and some speculation about the places in the learning process at which each type of system might be appropriate.

5.2 *Classification of Educational Computing Systems*

Many authors have sought to classify the uses of computers in education. For example, Bork (1979) proposed a scheme based on the *amount* of interaction a program engendered between student and computer. Others have focused on the *nature* of the part played by the computer in any interaction which takes place: Taylor (1980) identifies three different roles for the computer: Tutor, where the computer presents subject material to the student, Tool, where the computer is used by the student to assist in a task, and Tutee, where the student programs the computer to carry out a task. Papert (1987a) echoes this classification, seeing computers used first as "mechanized instructors" for delivering tutorials, then as "tools for doing something else: as calculators, word processors, simulators, or whatever", and finally as "microworlds", of which Turtle Geometry is an example, and where programming is central.

The field is also rich in acronyms for different types of software: CAI, CBT, CAL and so on. However, as Adams (1988) points out, "common usage has rendered many of these terms useless, since different authors will assign quite different meanings to the more popular terms." Furthermore, in Adams' opinion the classifications of these acronyms "tend to be about technical aspects of software and hardware; what Papert regards as technocratic values." Adams clearly favours categorisation based on educational values.

Another distinction can be made between educational computing systems which include an Artificial Intelligence derived *tutor*, and those which do not. Here, tutor has a different meaning from Taylor's use above. What is meant is a piece of software which attempts to provide the student with some of the things a human teacher would normally offer, such as guidance, hints, thought provoking comments, and assessment of progress. The tutor runs alongside the software providing the subject material (in practice the two components are often tightly intertwined) and maintains a dialogue with the student. The construction of such Intelligent Tutoring Systems is a highly technical field, drawing on research from many different subjects including artificial intelligence, cognitive science, knowledge engineering, and human computer interaction. Surveys of the techniques involved may be found in several texts, for instance: (Sleeman and Brown, 1982; Wenger, 1987; Frasson and Gauthier, 1990).

Elsom-Cook (1990) identifies a class of Intelligent Tutoring Systems which are especially relevant to objectworlds. He describes a new paradigm for educational computing systems called Guided Discovery Tutoring. This brings together Discovery Learning Environments and Intelligent Tutors to provide a system in which students are free to explore a subject domain, while the tutor provides guidance. Figure 5.1 illustrates the concept.

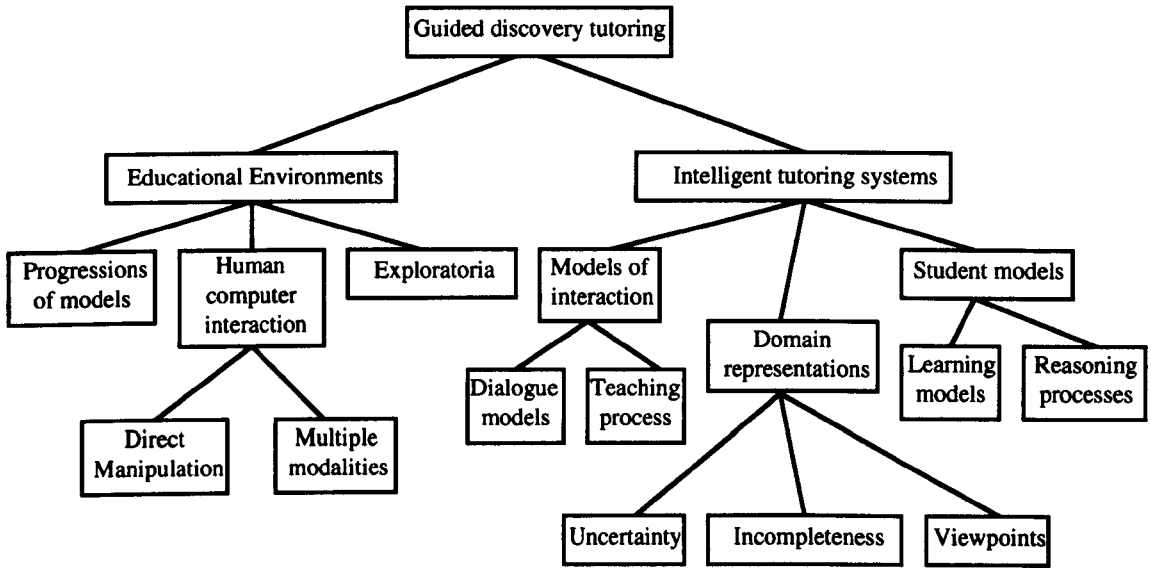


Figure 5.1 Guided Discovery Tutoring (after Elsom-Cook (1990) p11).

In the sections which follow we will describe a number of well known educational computing systems, some with tutor components, some without. What all the systems we will cover have in common is that they support discovery learning in their domain of interest. Elsom-Cook's synthesis gives us some justification for treating the learning environments separately from the tutors and, following Adams, we will emphasise the educational differences between systems, rather than the technical details. We will also try to avoid the use of acronyms, preferring terms with a more definite meaning. In fact we will restrict our comparison of objectworlds to two other classes of program: Simulations and Modelling Systems. We intend to show that these are the closest relatives objectworlds have in the range of educational software, yet they still offer very different experiences to learners.

However, before we go on to describe actual systems we should clarify our own use of terms. From our point of view, Modelling Systems, Simulations and objectworlds are closely related because they all exploit computer based models to convey their educational message. What differs is the way these models are *used*. Although in many ways the differences between the three systems are a matter of degree and the lines of demarcation

are somewhat indefinite, nevertheless we believe some general distinctions can be drawn.

In a Modelling System, the educational activity involves students actually constructing a model by some means. The facilities offered for model construction vary widely. Some systems offer programming languages, others allow the user to draw symbolic diagrams which represent the model and its parameters or initial values. A third technique allows the user to work by directly manipulating graphical components on the screen to build a model. Whatever the method of construction, the computer converts the model into an executable form and runs it, generating displays of its results so that the user may inspect its behaviour. Far from finishing at this point, many systems encourage the user to go back and explore the consequences of adjustments to their model.

In contrast, most Simulations conceal their model from the learner and simply offer an interface which allows them to vary parameters and observe results. The focus is on the interpretation of these results and their comparison with reality. As Brna (1991) puts it:

> "...simulations tend to emphasise the issue of the simulation's fidelity to the real world whereas modelling emphasises the exploration of the consequences of the model." (Brna, 1991)

Although we have drawn a sharp line between Modelling Systems and Simulations, in actual use things can become a little blurred. A Modelling System might be used just in its execute mode, with a preset and unexamined model, thus shifting the emphasis onto the results; a Simulation could be used purely to demonstrate how a particular model behaves. In practice however, the design of systems tends to encourage the interaction styles described above, as the next two sections (which describe well known exemplars for each category) will illustrate.

5.3 *Simulations*

Once a model for a physical process or a state of affairs exists on a computer it can be executed and observed while a range of initial conditions or parameters are tried. When a reliable model is parcelled into a friendly interface, which allows the user to vary the inputs and view results with relative ease, then we call the system a *simulation*.

5.3.1 *SOPHIE*

One of the earliest Intelligent Tutoring Systems was the SOPHIE program (Brown, Burton and Bell, 1975) which combined a tutor with a circuit simulator to give a system that lets students practice their electronic circuit debugging skills. The tutor is novel in that it maintains an English language dialogue with the learner. Students can ask SOPHIE questions like "What is the voltage across R23?" simply by typing them on the keyboard. Similarly, SOPHIE can provide advice in English, such as "The base current of Q4 seems to be incorrect." The educational intention of SOPHIE is actually to give students the opportunity to fix a faulty simulated power supply unit. They are free to vary the parameters of the model in the sense that they can set component values. The output from the model is observed by making "measurements" of voltages and currents in question form, as above. SOPHIE has been the subject of a great deal of research and has been used with success in real classrooms. The researchers devised a game in which one user inserts a fault in the power supply which a second user must find.

> "The game was designed with two instructional goals in mind. First, we wanted a self-motivating activity that promoted cost-effective troubleshooting. Second, we wanted an activity that required the student to exercise his causal and teleological understanding of the device." (Brown, Burton and De Kleer, 1982)

The underlying environment of SOPHIE is a simulation. SOPHIE is not an objectworld firstly because there is no continuously visible object in the sense we have developed in this thesis: learners refer to a printed schematic which depicts the *configuration* of the circuit but not, for the most part, its *state*. Indeed, SOPHIE must expressly conceal the state of circuit components or else the central activity of finding faulty items would become trivial. Secondly, the language of interaction, although flexible and robust, is a query language rather than a programming language. Figure 5.2 summarises SOPHIE's status.

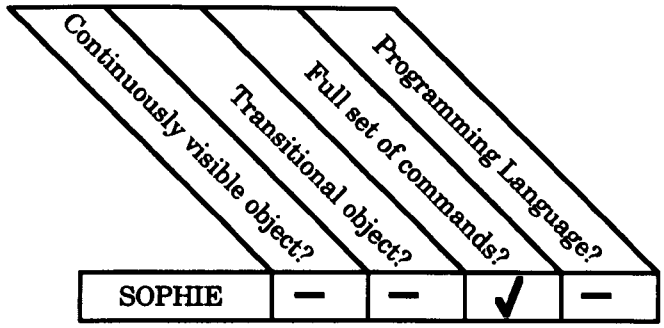


Figure 5.2 SOPHIE.

Nor is SOPHIE a modelling system, since ordinary users cannot alter the circuits (that task is reserved for system programmers) or build their own. Learners can, however, use commands to set and inspect certain circuit parameters. Essentially, learners are being given the opportunity to build their own detailed mental model of the device and its fault modes.

5.3.2 *STEAMER*

Hollan, Hutchins and Weizman (1984) describe STEAMER, a simulation of the steam propulsion unit of a large ship. Such units are highly complex and to aid novice engineers the authors have built an "interactive inspectable simulation" with a sophisticated graphical interface. The physical machinery is represented schematically on the screen and it is clear that the objectworld paradigm, with its requirement that all the objects (and their state) are continuously depicted, would cause an information overload. Furthermore, the objects in STEAMER are so numerous (and disparate) that manipulating and inspecting them via the objectworld strategy of operators and functions would be unwieldy. The simulation approach is obviously the right one for this domain.

However, there is an addition to STEAMER, the *feedback minilab*, which lets the learner study components of the system in isolation. Within the minilab objects and their state *are* continuously depicted and the direct manipulation interface allows the inspection and manipulation of parameters. But these devices are highly specific machines such as pumps and valves, and the system defines allowable, *correct*, ways to use them. They cannot be considered transitional objects in the sense we established in chapter two.

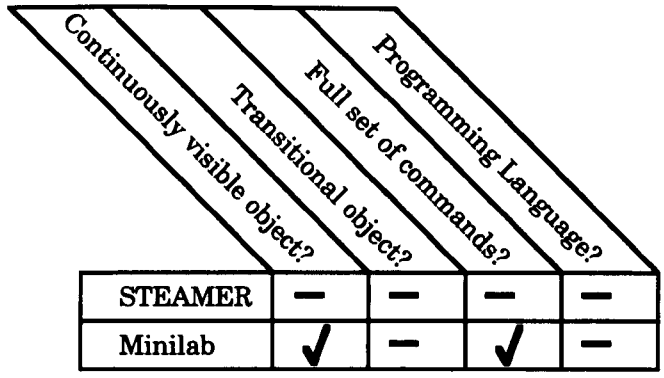


Figure 5.3 STEAMER and the Feedback minilab.

Hollan, Hutchins and Weizman view the kinds of learning fostered in the two environments as complementary: both are intended to help learners form mental models of complex devices, but on different scales. Both the simulation and the minilab contain limited tutorial capabilities to support this aim.

5.3.3 SMITHTOWN

Another Intelligent Tutoring System based on a simulation is SMITHTOWN (Shute and Bonar, 1986; Shute and Glaser, 1990). This program presents a hypothetical town whose economic details (such as population size, consumer preferences, average income, state of markets etc.) are displayed continuously on the screen. These details can all be inspected and set by the student. In these respects the system is conforming with our definition, and in fact the authors refer to it interchangeably as a simulation and a microworld. Nevertheless, as figure 5.4 shows, SMITHTOWN is in our terms a simulation. First of all, the things displayed on the screen are the state of the town and not the town itself (or even some stylised depiction of it). Secondly there is no programming language made available to users - they set and inspect the state of the town via the program's graphical interface.

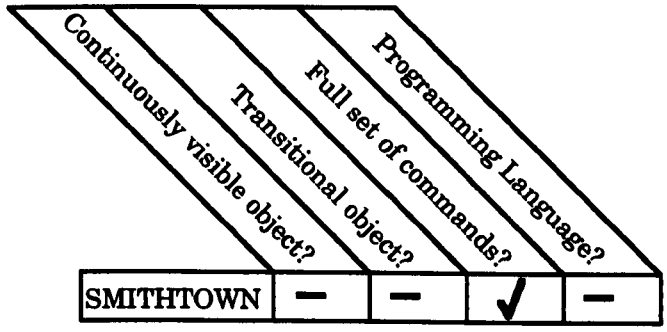


Figure 5.4 SMITHTOWN and the objectworld criteria.

The designers' choice of the simulation paradigm is justified by the types of learning they wish to promote. For instance, they want students to be able to *discover* formal concepts, such as the law of supply and demand, which are embodied in the system. Indeed, the tutor component of SMITHTOWN is intended to guide students to just this sort of discovery.

> "SMITHTOWN is a highly interactive program, allowing students to pose questions and conduct experiments, testing, and enriching their knowledge bases of functional relationships by manipulating various economic factors." (Shute and Glaser, 1990)

Students are not expected to construct a model of the law, as they would in a modelling system.

5.3.4 *The Alternate Reality Kit*

We briefly mentioned the Alternate Reality Kit (Smith, 1986,1987) in chapter two. ARK was designed and built by Randall Smith at the Xerox Palo Alto Research Centre. He describes it as "a system for creating interactive animated simulations." ARK consists of a large set of graphical objects such as switches, sliders, buttons and meters. The objects are continuously visible on the screen and may be manipulated directly with the mouse. All objects can be given a velocity and ARK animates them smoothly across the screen. A special class of objects, called *interactors*, allow behaviours such as gravity, friction and collision detection to be defined between objects. With these building blocks it is possible to construct a wide range of simulated experiments. Smith has built projectile launchers which show objects moving under the force of gravity. Other applications built in ARK include a simulated bubble chamber, for observing the paths of charged particles, and a factory manufacturing soft drinks.

However, while ARK's friendly interface makes *using* these simulations very straightforward, actually constructing them is very hard. Smith envisages two classes of user: "The *applications-level* user might typically be a student carrying out a simulated lab. At a lower level the *simulation builder* is the creator of a particular application. There may be a role for another layer below that, populated by individuals who create tools for use by simulation builders" (Smith, 1987). In practice, most users of ARK work at the applications level, using exactly what they are given. To be a simulation builder, the user must be

able to program in ARK's implementation language, Smalltalk-80, and needs to know a good deal about ARK's internal mechanisms. Smith reports having observed 50 application users against two simulation builders.

From an educational point of view then, ARK is what an applications level user sees - a simulation. We should stress again that ARK simulations are straightforward to use since they employ familiar metaphors. A parameter, such as velocity, can be "attached" to a slider control and varied with the mouse. Gravity may be turned on or off with a switch. Once again, the emphasis is on leading the user to achieve an understanding of a concept, such as one of Newton's laws of motion.

The ARK simulation builder though, has access to the programming language Smalltalk. Therefore, by our definition it could be said that a simulation builder is working in an objectworld. ARK's objects are continuously visible (if they haven't fallen off the screen!), they may be designed to embody some important concept (for example, Smith has built Newtonian particles), and the programming language can certainly support analogues of the command sets required by the definition. Nor would it be too difficult to build syntonic commands (see chapters 1 and 2) in ARK and create full blown transitional objects. Figure 5.5 summarises the situation for the two classes of user.

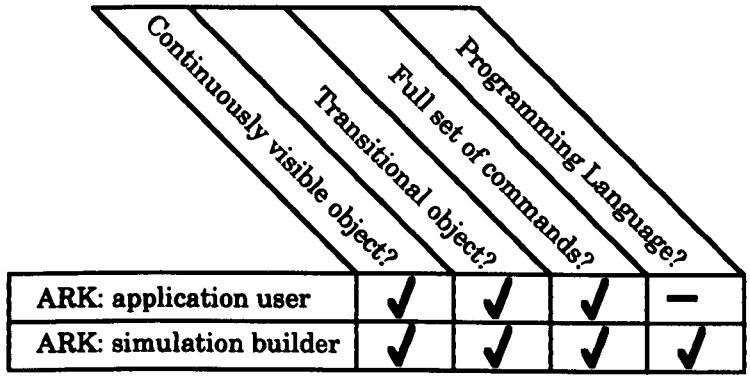


Figure 5.5 ARK and the objectworld criteria.

5.3.5 *NEWTON*

Another system we wish to discuss is based, like Gravitas, on simulations of Newtonian objects. NEWTON (Teodoro, 1990) allows learners to subject particles to forces and observe the affects on their motion as they move across the screen. Unlike Gravitas, the number of particles is limited to two and nor do they interact with each other gravitationally. However, as with ARK,

friction and gravity (as a simple acceleration acting down the screen) may be turned on so that it is quite easy to observe the trajectories of 'real' projectiles. Other features allow the user to display the velocity vectors of the particles and plot graphs of chosen variables. Any run of the simulation may be replayed at will making the detailed examination of particle motions straightforward.

NEWTON's particles are transitional objects because they are simplified versions of the formal abstractions physicists call Newtonian particles, and they are continuously visible. The system's control panel, shown on the left of figure 5.6, also contains buttons which accelerate a particle in one of eight directions. These force buttons are comparable to the syntonic functions of Gravitas and Turtle Geometry.

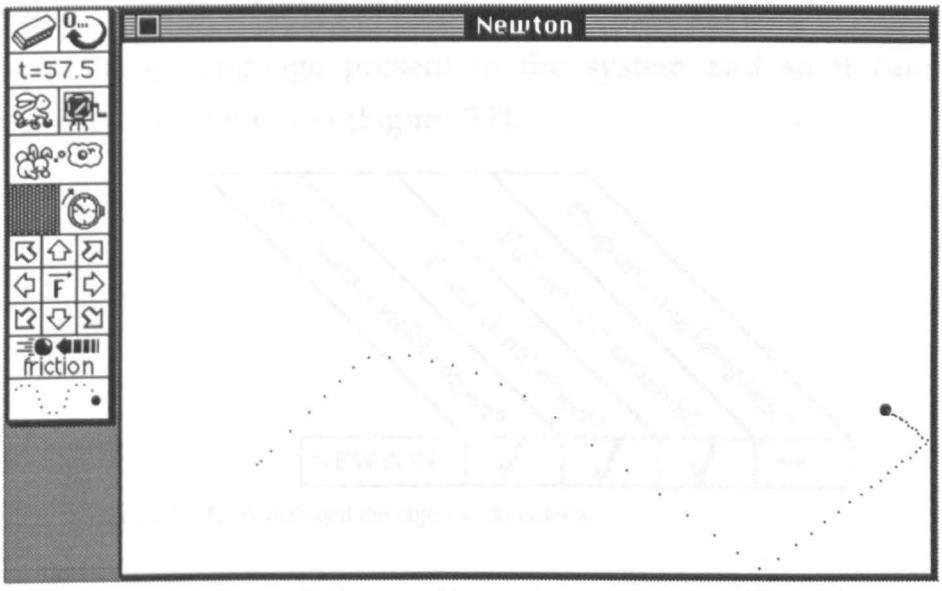


Figure 5.6 NEWTON showing the control panel and a single particle with friction.

Some of the designer's comments reinforce this view of NEWTON's particles as transitional objects. Certainly, the designers of the system seem to have similar intentions to Papert in the sense that they wish to create new computational objects for education to use:

> "NEWTON is intended to extend the range of manipulable objects in the learning of dynamics, namely the abstract physical concepts of velocity, force, momentum, energy, etc. It also allows the student to confront multiple representations of movement, in real time." (Teodoro, 1990)

However, in contrast to Gravitas, the user must take account of a particle's mass to understand the effect of a force, and they have to select a direction for it. As we mentioned in section 2.8 of chapter 2, Gravitas' syntonic commands

always act in (or perpendicular to) a Massob's direction of travel and the accelerations are independent of mass. In other words, manipulating NEWTON's particles is more cognitively expensive than manipulating Massobs. On the other hand, NEWTON's particles have a less complex behaviour since they do not gravitate with each other. If we assume that the basic cognitive effort required to work with these kinds of object is a function of their inherent complexity and the intricacy of their syntonic functions, then particles and Massobs are probably on a par, and both a little more taxing than Turtles. We should caution against reading too much into this assessment though, as it does not compare the cost of doing *educationally valuable* things with the objects.

Even if we accept NEWTON's particles as transitional objects, still there is no programming language present in the system and so it cannot be an objectworld by our definition (Figure 5.7).

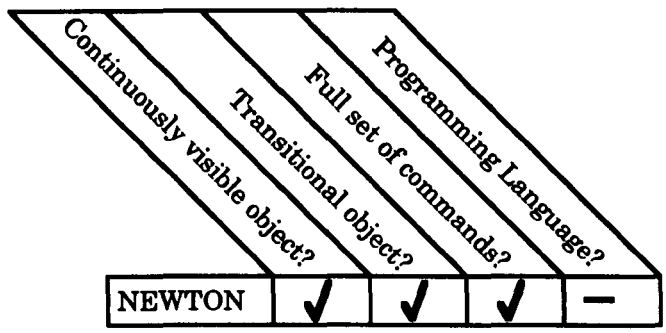


Figure 5.7 NEWTON and the objectworld criteria.

The constructive possibilities offered by something like a Logo interpreter are therefore not present, but the designers seem to have different goals anyway:

> "One of the assumptions in the design of NEWTON is that knowing and understanding means 'becoming familiar with different representations of a phenomenon'. With the program students can have the possibility to become familiar with the effects of forces, with vectors that represent physical constructs, and time (either in an analogical graphical representation or property vs. time graph), etc." (Teodoro, 1990)

On the other hand, it does seem clear that the addition of a programming facility could enhance NEWTON. After all, the particles are very similar to Massobs with their gravitation turned off (a situation accomplished in Gravitas by setting the universal constant of gravitation to zero). Furthermore,

NEWTON's control panel is similar in concept to Gravitas' buttons and displays, and so it is reasonable to assume that the same synergy between graphical and programming interfaces that we discussed in chapter three would be generated.

5.3.6 ROCKET

The last system we wish discuss in this section is also designed to offer learners experience of Newtonian objects. Brna (1989) describes ROCKET, a re-implementation and refinement of Dynaturtles (diSessa, 1982). Like diSessa, Brna is interested in the idea that such systems can be lead students to confront their misconceptions about Newtonian dynamics. ROCKET allows the investigation of the motion of a body moving in two dimensions, free of friction or gravitational forces. The body may be given 'kicks' (or instantaneous velocity increments) of variable magnitude and the direction of the kicks may also be varied by the learner. Like Dynaturtles, the user can drive ROCKET interactively from the keyboard: a press of the *L* key rotates the rocket's kick heading (indicated by an arrow) 10 degrees to the left. The *R* key rotates it right by the same amount. Pressing a number from 1 to 9 applies a kick of that magnitude to the rocket (the units are arbitrary).

The refinement added by Brna is a simple programming language which allows the learner to pre-compose sequences of commands, then execute them to try to achieve some goal. The point behind this addition is to make it easier for a teacher to infer the students' plans as they try to carry out their tasks. Brna contends that the purely interactive modality of TARGET (the name used by diSessa for his Dynaturtle game) is too narrow a channel through which to view their intent:

> "The language used by the students to communicate with the computer in TARGET is so impoverished that it becomes very difficult to infer student's plans and strategies reliably. I believe it is an improvement to provide a simple programming language which allows the student to devise and communicate game-playing strategies - whether or not a student has Newtonian conceptions about dynamics." (Brna, 1989 p30)

In classroom trials Brna made extensive observations of students using ROCKET in both interactive and programmed modes as they tried to hit a target. (Figure 5.8)

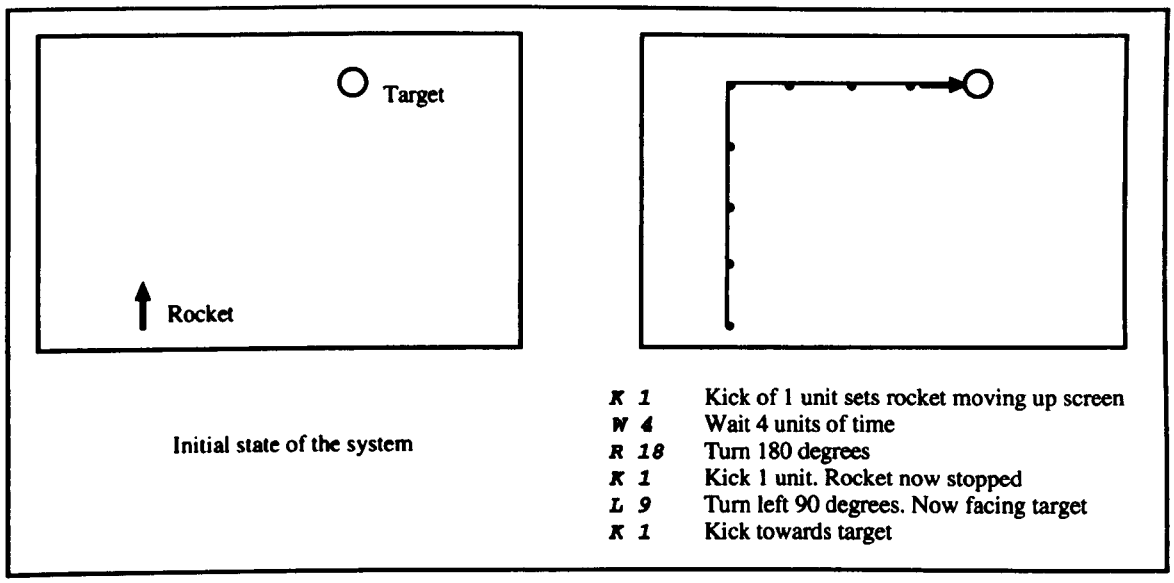


Figure 5.8 A task and a program in ROCKET

Brna was especially interested in diSessa's idea of a *learning path chart* for such problems. This chart is supposed to describe the possible paths a student make take in their attempts to solve a particular task. Brna wished to:

> "...examine the claim that the game can be used in the classroom because the *learning path* chart provides a sensible basis for a small part of the physics curriculum. The justification for this is that the learning paths allow for a 'natural' development of the ideas needed for mastering the physics associated with the game." (Brna, 89 p29)

Brna observed many strategies as his students set about the task but he concluded that it was difficult to reliably infer that they were confronting their own mistaken beliefs as a result of using the program. The *Newtonian corner* strategy (see section 1.10 and figure 5.8 above), for instance, is equally understandable from both Newtonian and Aristotelian points of view. He notes that the addition of the simple programming language does make it easier to diagnose students' misconceptions but still does not yield sufficient reliability. Brna suggests that one way forward would be to add a more powerful programming language, such as Logo or Smalltalk, with built in primitives that better reflect the problem domain:

> "My conclusion is that the current language used for ROCKET is too closely equivalent to *machine code* to be useful. A better approach is to provide a higher level language. An even more promising approach is to provide a language which can be used by the students themselves to describe their beliefs about the underlying physics." (Brna, 89 p29)

Since there are inevitably occasions where the teacher is attempting to infer the depth of an objectworld user's understanding, Brna's comment has great relevance to Gravitas and we will return to it at the end of this chapter (in section 5.5). For the time being we note that ROCKET, interesting extension of the simulation idea that it is, still does not fall into the objectworld category. It does not implement a full set of state operators for the central object (after all, a position operator would render the game trivial) and its programming language is too restricted.

5.4 *Modelling Systems*

Modelling is without doubt one of the most important applications of computers. As Howe et al (1979) point out "Perhaps the most versatile modelling system is the digital computer." Governments use computer-based models of the national economy; commercial operations model their finances and production lines; engineers model bridges, engines, pipelines, and other structures. Computer models of political and military conflicts have been built and there are even models of legal systems, which can generate decisions on actual court cases. The wide range of possible applications has led some authors to try to capture what is common to the various kinds of model:

> "A model is a representation of structure. There are many kinds of models: a model might be a physical object, or it might be a structure of related ideas, which might be expressed informally in words or diagrams, or which might be expressed more formally." (Mellar, 1989)

Schecker expresses the same idea in a slightly different way, defining model building systems as:

> "Context-free software tools that support the user in representing a part of the 'touch-and-show-reality' in the form of an abstract, quantifiable system of parameters and their relationships (the *model*), which predicts the behaviour of the real system." (Schecker, 1990)

Models can be expressed in different ways: Perhaps as a set of mathematical equations or a set of logical relationships. In some systems the medium of expression is a programming language of some kind. Other systems, as we will see, allow models to be built graphically, as diagrams or schematics.

From an educational point of view modelling engages learners in two valuable activities: first of all they must describe a process or state of affairs in a form that the computer can evaluate. Once this is done they can turn their attention onto the behaviour of the model, its predictions and limitations. As we indicated in the previous section, at this point the learner is often led into a cyclic process of modifying the model and observing the effect the modifications have on the results.

The case for the inclusion of modelling activities in the school science curriculum has been made in, for instance, Oke and Jones (1982), and Millwood and Stevens (1989). As the latter point out, modelling is what working scientists spend much of their time doing. In the sections which follow we will describe a few of the better known modelling systems.

5.4.1 *The Dynamic Modelling System*

The Dynamic Modelling System, DMS, (Ogborn and Wong, 1984) was developed for use on school microcomputers of the early to mid 1980s. It is designed to deal with dynamical models - "that is, models which compute the evolution of a system step by step. The British economy, a satellite in orbit or the rabbit population on an island are all examples of systems for which such a model can be constructed" (Ogborn and Wong, 1984). DMS presents the user with an editor into which statements of the programming language Basic can be entered to form a model. A second editor lets the user assign initial values to variables in the model. The system will then loop around the model, computing new values for variables and plotting the results on a graph.

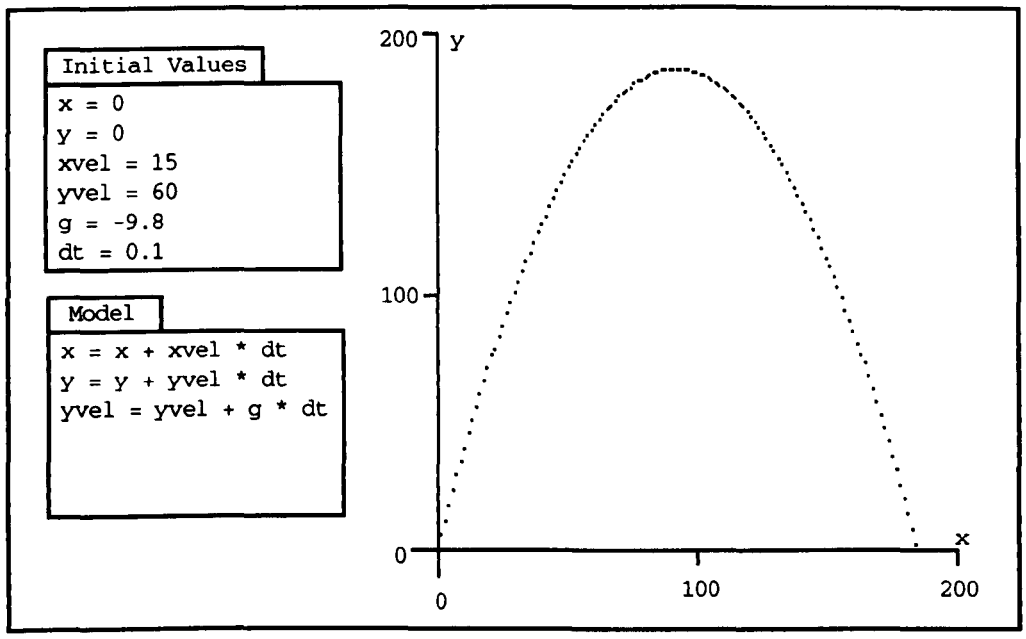


Figure 5.9 DMS being used to model projectile flight (This is a simulated image. In practice the Model, its Initial Values, and the results are viewed separately).

The system is easy to use and although the model is limited to twenty or so lines of Basic, some interesting topics can be investigated. However, if we examine DMS from the point of view established in chapter one, we see that it is not an objectworld. First of all, the learner's attention is focused on a set of equations and parameters rather than a continuously visible object. Nor, in the terms we

have set out, can tables and graphs be thought of as transitional objects. They *are* formal systems, rather than stepping stones to them.

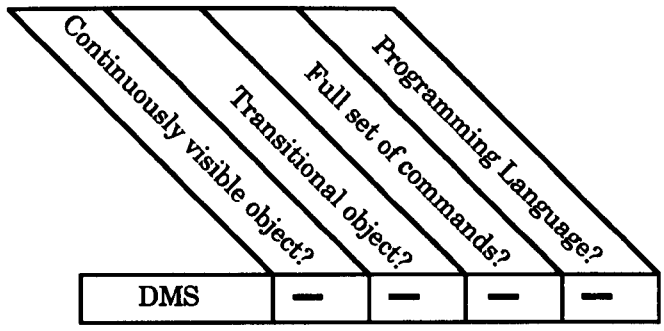


Figure 5.10 DMS and objectworld criteria.

Without objects the concept of simple commands to inspect and adjust attributes does not apply. Furthermore, the notation in which the models are expressed is really too weak, most especially in its support of data types and control structures, to be considered an objectworld language. Figure 5.10 illustrates the state of affairs.

Such a list of negatives is almost certain to give the impression we are unimpressed by DMS. However, this is far from the truth. We are simply concerned to stress the differences between objectworlds and other systems.

5.4.2 STELLA

Another system dealing with dynamic models is STELLA (Richmond et al, 1987). Here the user can build a model on the computer screen by assembling a collection of graphical *components*. The kinds of components available in STELLA are called stocks, flows, converters and connectors, and the program is based on the observation that a large class of models can be represented by systems of these objects. The user constructs a diagram from the basic components and sets initial values for the stocks and flows. Then, equations can be entered to define the detailed behaviour of components. Schecker points out that this two step process can lead to an advantage for the learner:

> "Icon-oriented systems like STELLA force the students to engage in a qualitative, principle oriented analysis of the problem before they can work on the equation level. Prior to a definition of special functional relationships the relevant quantities have to be defined and the structure, i.e. the conceptual features of the model, must be formulated. The students are thus introduced to the strategy of expert solvers." (Schecker, 1990)

Once the model is complete, STELLA can be asked to produce graphs of selected quantities while the model is iterated over an independent variable, such as time. Sophisticated models can be constructed in STELLA, and its solving algorithm can be set to give almost any degree of accuracy. However, it is important to note that although the basic components can be thought of as objects, they often require users to think in terms of abstractions; for example, the learner must adjust to the idea of representing distance as a stock.

In this sense, STELLA is not truly context-free. Its basic metaphors of flows and stocks angle it towards certain kinds of problem. As Schecker puts it, the system suffers from some "constraints for physics and chemistry which originate from STELLA's orientation towards the social and economic sciences" (Schecker, 1990). Nevertheless, in the same paper Schecker does show STELLA being used to build working models in physics, such as the diagram of figure 5.11, representing a restricted form of planetary motion (the case where planetary mass is insignificant with respect to the mass of the Sun).

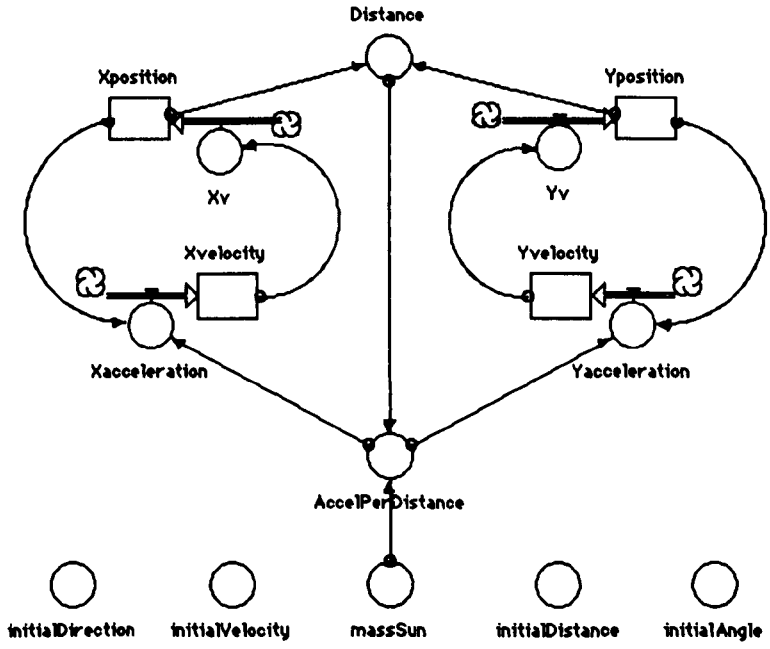


Figure 5.11 STELLA model of planetary motion (After Schecker, 1990).

Schecker discusses the value of computer based modelling for students, claiming that it "supports the user in an active process of applying mental models formed in his mind to a new phenomenon. With the help of a modelling system the modeller explores the consequences of his conceptual assumptions about the phenomenon" (Schecker, 1990). It is implicit in this view that learners need to have *some* formal understanding of the domain, at

least to a level that allows them to express ideas as algorithms or diagrams that the system can execute.

Considering STELLA against our criteria, it is clearly not an objectworld. As we said above, the flows, stocks and converters are continuously visible, but they require the user to think in terms of abstractions. This reduces the likelihood of them connecting to a novice's concrete experience and therefore makes them less suitable as transitional objects. On the other hand, STELLA has facilities functionally similar to Turtle or Massob commands, which make it easy to examine or alter the values of the objects. STELLA's notation, like that of DMS, is weak, being tuned to the expression of difference equations.

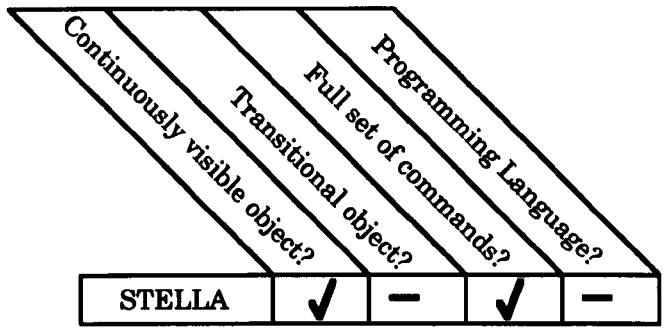


Figure 5.12 STELLA and the objectworld criteria.

However, it is interesting to note that STELLA's facility to combine graphical components into systems by direct manipulation is in some ways a substitute for the control constructs and procedure definitions of conventional languages.

5.4.3 IQON

The next system we will examine tries to marry modelling and qualitative (or more strictly, semi-quantitative) reasoning. In IQON (Miller et al, 1990), as in STELLA, the learner uses a Direct Manipulation interface to build a diagrammatic model of a real-world system. The difference is that IQON's models represent the *qualitative* relationships between special objects called continuous-valued variables, or boxes for short.

The level in a particular box is affected by the level in others through positive and negative *links*. A positive link implies that a rise in the source box will produce a rise in the target box, while a negative link implies the reverse. An underlying mathematical engine ensures that the model is normalised, in the sense that the level in all boxes is constrained to lie between

plus and minus one. IQON can plot graphs of the levels in the boxes over time.

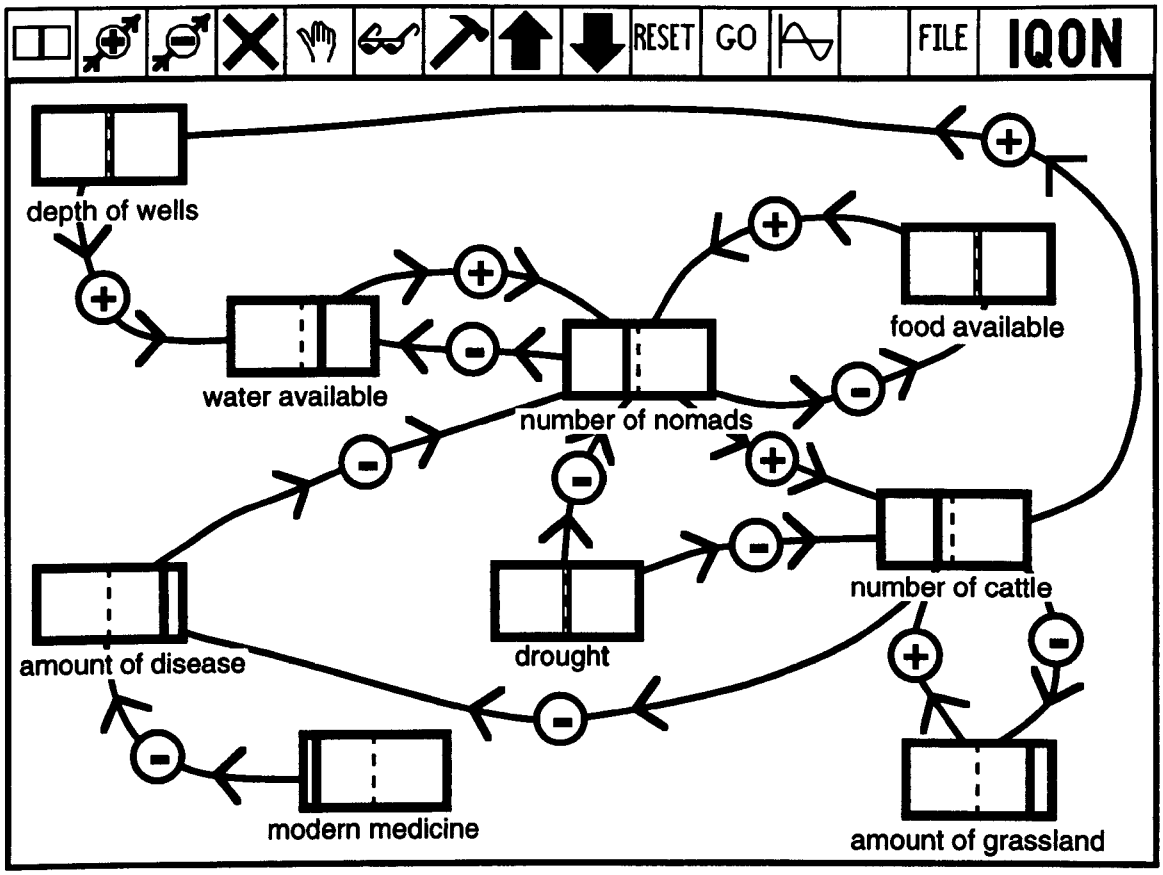


Figure 5.13 An IQON model of the Sahel (After Miller et al, 1990).

For instance, there is a qualitative relationship between food and population: IF there is more food present THEN a larger population can be supported. If we add another variable, the complexity increases rapidly: IF the population increases THEN more water is consumed THEREFORE less water is left for agriculture. Nothing needs to be said about absolute values. Complex models can be built up in IQON and the authors report using IQON to simulate a wide range of real world situations: queue length at supermarkets, the atmospheric carbon cycle, illegal drug trafficking, and so on. They also give backing to the qualitative approach in education by pointing out that examples of qualitative thinking abound in the history of science.

To use boxes and links to stand for different things requires a student to *abstract* the concepts of interest and to do this they must already have some level of formal understanding. The boxes and links are not, therefore, transitional objects although they are easy to examine and alter. IQON does not offer the user a programming language, but as it was developed in Smalltalk (Goldberg and Robson, 1983) this might be possible to arrange. So, as figure 5.14

shows, IQON is not an objectworld. However, compared to STELLA, the mechanisms which allow the computer to *execute* the user's graphically constructed model are more deeply hidden.

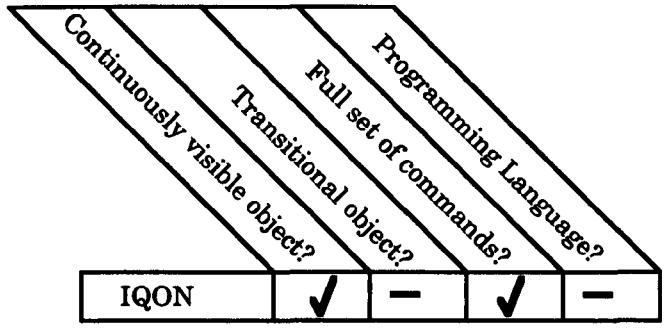


Figure 5.14 IQON and the objectworld criteria.

The use of IQON (and other systems) in real classrooms is reported in (Bliss et al, 1992). Some of the researchers' main findings are concerned with the ways in which children decide on the entities that will be *variables* for a given problem. They also find that when building models, children often invent other entities which do not really correspond to continuous-valued variables but which are represented as such because IQON offers no other way. Bliss classifies these extra entities into three types: objects, events and processes, according to the role the learners ascribe to them. With regard to *links* between variables, Bliss has the following to say:

> "The pupil's description of links or relations can be either causal or non-causal, with causal links being seen as one variable having some sort of influence on another, whilst non-causal links are just co-occurrences of high or low values" (Bliss et al, 1992)

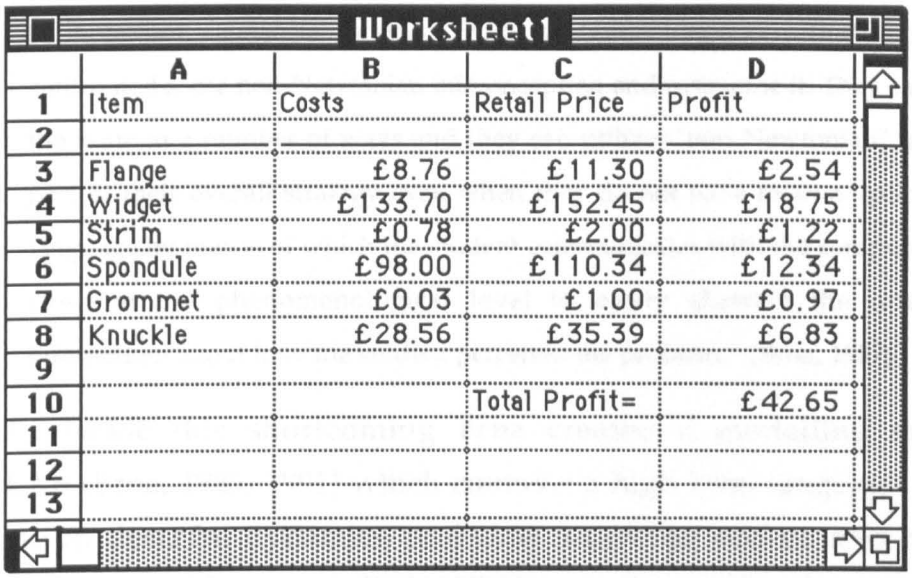
As we said above, variables and links are *abstractions* which can stand for very different kinds of real world situation, and to use them requires some formal understanding of the concepts they represent. In addition to this fundamental hurdle, Bliss reports that many children find it hard to cope with models as a whole. These two points suggest that a more appropriate role for modelling may be in the later stages of concept acquisition.

5.4.4 *Spreadsheets*

Another class of modelling system is the computer spreadsheet package. These are commonly used by companies (or individuals) to model stocks, production and cashflows and to carry out projections of costs, turnover, and profits. To do this, data is entered into *cells* on a grid and relationships between

them are set up. For instance a cell may be defined to be the sum of the column of cells above it, as is the case for cell D10 in figure 5.15.

Although the facilities of spreadsheets are highly tuned to their primary purpose - the construction of an active data structure - several authors have reported positively on their teaching uses in topics as diverse as chemistry (Brosnan, 1990), geology (Holm, 1988), and mathematics (Eyler, 1990; Galizia, 1990).



	A	B	C	D
1	Item	Costs	Retail Price	Profit
2				
3	Flange	£8.76	£11.30	£2.54
4	Widget	£133.70	£152.45	£18.75
5	Strim	£0.78	£2.00	£1.22
6	Spondule	£98.00	£110.34	£12.34
7	Grommet	£0.03	£1.00	£0.97
8	Knuckle	£28.56	£35.39	£6.83
9				
10			Total Profit=	£42.65
11				
12				
13				

Figure 5.15 A typical spreadsheet.

Most recent spreadsheets have considerable power in terms of both basic operations and graphical features. In fact, Microsoft's Excel contains a command language which is a version of Basic extended with data types corresponding to groups of cells. By attaching commands to sequences of cells programs can be constructed. It is even possible to group commands together and refer to them by name. Furthermore, Excel has considerable graphical power for displaying results.

To date the reports in the educational literature focus on topics which fit the spreadsheet paradigm, i.e. the construction of an active structure. Chemical lattices, rock strata, and mathematical series are examples. To our knowledge, no one has used a spreadsheet to construct a transitional object. Such a development cannot, however, be ruled out, and spreadsheets remain an interesting class of system.

5.4.5 DYNLAB

In section 5.3.6 we noted Brna's conclusion that a simulation, even one augmented with a simple programming language, is not an ideal tool for forcing students to confront their misconceptions about the dynamics of Newtonian bodies. He points to two examples – TARGET (diSessa, 1982) and a variant of that system – ROCKET (Brna, 1989) which both simulate a body moving in a two dimensional space free of friction or gravity, and comments:

> "...it cannot easily be inferred from students' behaviour that they have confronted some non-Newtonian misconception and overcome it. They can evade the issue in a number of ways and they can utilize "non-Newtonian" tactics as part of their overall strategy even when they do not have misconceptions. The language by means of which the student communicates with the computer is too close to the phenomenological level to easily abstract the interesting information about how the student perceives the problem." (Brna, 1987)

To tackle this shortcoming Brna created a modelling system called DYNLAB (Brna, 1989; 1991) which contains a high level language that students can use to describe dynamics problems involving one or two bodies. Users model these problems (or SITUATIONs as Brna calls them and which he presented to students on worksheets) in three stages: First they use the language to define a MAP – a description of the territory over which the body is to move. Secondly they describe the JOURNEY the object is to make, either in terms of a set of constraints or as a set of interesting events. Finally, the students are required to describe the FORCE that is needed to drive the object on its JOURNEY around the MAP.

Brna gives an example use of the language to model an icecube sliding along a tabletop. Figure 5.16 shows the layout.

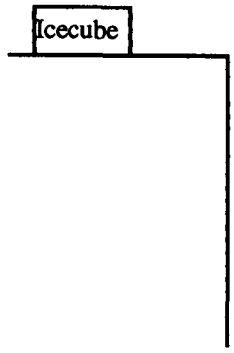


Figure 5.16 The SITUATION to be modelled

A model of this SITUATION begins with the MAP definition:

```
SLIDE:MAP:TABLETOP
DISPLACEMENT  BEGIN   EDGE    10M  90
DISPLACEMENT  EDGE    FLOOR   20M  180
JOIN          BEGIN   EDGE
JOIN          EDGE    FLOOR
```

This definition is to be read as follows: (i) Begin the description of the MAP for a SITUATION named TABLETOP. (ii) Two points in the MAP, called BEGIN and EDGE, are separated by a distance of 10 metres along a bearing of 90 degrees. (iii) EDGE and a third point, FLOOR are separated by 20 metres along a bearing of 180 degrees.

The next step is the JOURNEY definition:

```
SLIDE:JOURNEY:ICECUBE
START         BEGIN
MASS          2KG
VELOCITY      BEGIN   2M/S    90
```

This may be read as: (i) In the SITUATION named SLIDE a JOURNEY is to be made by the object called ICECUBE. (ii) The START of the JOURNEY is at the point called BEGIN. (iii) The MASS of the ICECUBE is 2KG. (iv) The VELOCITY of the ICECUBE at the point BEGIN is 2M/S along a heading of 90 degrees.

The final stage of the model is the definition of the FORCEs that act on the bodies in the SITUATION. There is only one body here but two forces must be defined:

```
SLIDE:FORCE:WEIGHT
ACTS          ICECUBE
FORCE         ONE     19.6N   180

SLIDE:FORCE:REACTION
ACTS          ICECUBE
FORCE         ONE     19.6N   0
DISPLACEMENT  10M
```

These definitions state that (i) a force of 19.6 Newtons (ICECUBE's mass times the acceleration due to gravity) acts on ICECUBE along a bearing of 180 degrees i.e. downwards. (ii) a force of 19.6 Newtons acts on ICECUBE along a

bearing of zero degrees (i.e. upwards – this is the reaction of the tabletop to the object) but only over a displacement of 10 metres.

Figure 5.17 illustrates how DYNLAB would run the SITUATION:

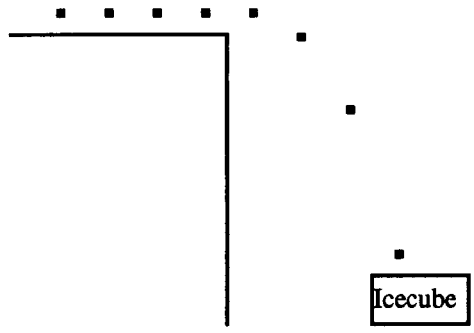


Figure 5.17 The SITUATION being executed

Brna used DYNLAB with a number of school students, setting them problems selected from the literature on children's misconceptions in dynamics. He notes that students were faced with conflicts between their expectations of behaviour and what DYNLAB produced:

> "Confrontations occurred and were resolved satisfactorily on a number of occasions. Those who took advantage of these confrontations were often the students who were *eventually* able to articulate their own beliefs. Encouragement of reflective thinking is one of the proposed advantages associated with the modelling approach." (Brna, 1987 p373)

Brna also reports positively on the utility of the language with respect to one of his original aims – the widening of the channel between student and system to improve the teacher's chances of inferring interesting things about the student's understanding:

> "It was found that the use of DYNLAB had advantages over TARGET or ROCKET. In achieving a goal such as that posed for students using ROCKET... [see section 5.3.6] ... students often appeared to be debugging non-Newtonian beliefs. It proved much easier to discriminate between students with Newtonian beliefs making use of sensible problem solving strategies and students with some misconception." (Brna, 1987 p373)

So, Brna concludes that the students' efforts in learning the formalism for programming situations into DYNLAB is worthwhile. The benefits are that the student is encouraged to reflect more deeply on the problems and there is

an increased opportunity for the teacher to diagnose faulty understandings. Although DYNLAB is not an objectworld (it lacks a complete set of state operators for the objects created by the user, and its language, tuned to the description of SITUATIONs, is not sufficiently powerful) this conclusion does have great relevance for systems like Gravitas, where deeper reflection by the student and better diagnostic opportunity for the teacher are just as important as within a modelling context.

5.5 *Synthesis*

No system of categorisation is perfect, and no system produces exact divisions between the different kinds of educational software that have been developed. There may be programs which overlap the three areas identified here. If we have exaggerated the degree to which our categorisation is unambiguous, then it has been for the sake of one simple point: that computer models can make an important contribution to education but that the framework in which they are placed affords very different learning experiences.

This statement is no longer a mere platitude. By clarifying terms and describing examples we have reached a point where it is possible to make succinct statements about the character of each type of environment we have been considering, and the corresponding educational experiences they offer. We believe that placing these descriptions in close proximity will lead to a clarification of the roles each can play in education.

An objectworld is a system which instantiates on computers an object that can easily be apprehended by learners, which can be manipulated in intuitively natural ways, and which, through programming, can be a vehicle for the examination of rich conceptual areas. It is essential that the mechanisms which give the object its interesting behaviour are hidden from the user.

A simulation takes a model of some process or state and allows the learner to vary parameters and inputs, and observe results and outputs. It is not *necessary* for the learner to comprehend the workings of the model, nor is such comprehension ruled out.

A modelling system allows the user to capture a concept, process or state in a procedural algorithm or other executable notation, and run it on the computer to generate predictions. As far as education is concerned, there is value in both the construction and the execution of a model. It is essential that the learner has some understanding of the model *and* the mechanisms for its solution.

Looking at the three types of system in this way suggests a progression. We speculate that each kind of environment is appropriate to different stages of concept acquisition. An objectworld gives learners the opportunity to explore and play with novel entities, whose specially designed behaviour

makes them concept-rich. Syntonic commands such as those of Massobs and the Turtle allow the learners to work directly with the objects and build up their fund of what Polanyi calls the "tacit component of articulate knowledge" (Polanyi, 1962). Our subjects in the studies of chapter three were developing their tacit knowledge of how objects move in gravitational fields as they manoeuvred the rocket to the moon. Typically, to explain their actions, learners draw on expressions which are not strictly applicable to the situation. For instance, Joe wanted to boost the rocket "when it's running parallel to Earth".

A simulation allows the learner to explore the limits of the behaviour generated by a particular mechanism, allowing them to build their formal understanding of the domain. The student knows there is a formal model driving the system and that the understanding of this model is one of the goals. SMITHTOWN, for example, contains a model of the economic law of supply and demand, which the students are intended to discover (with help).

Finally, modelling systems give learners the chance use their formal understanding to construct their own behaviours, or at least see how they are generated. This process can lead students to refine their articulate knowledge as they fit the components of a model together, or as they try to map the basic objects of systems like STELLA and IQON onto their problem. They can then move on to see what their creation *predicts*. At this stage the educational experience becomes similar to the experience of a working scientist.

Despite these distinctions between the learning experiences afforded by different classes of system it is wise to bear in mind the general consideration (touched on in section 5.3.6) pointed out by Brna (1987, 1989): In any system there will be many occasions when the teacher needs to be able to infer things about the student's understanding. To assist this process we need a broad channel of communication between student and computer. Brna suggests that a programming language one way to make an improvement and that a higher level language, with primitives that match the problem domain, is still better. Gravitas, as we have seen, is attached to a full strength version of Logo and so meets the first part of Brna's suggestion. Furthermore, Logo, in conjunction with the full set of state operators for Massobs (and the space they inhabit), should make it possible to experiment with a meta-language along the lines of the second part of his comment (although we have not done this). In

particular, predicates such as `pointing_at_target?` and `pointing_at_right_angle_to_path?` (Brna, 1989 p39) would be straightforward to implement. This represents important theoretical support for the language requirement in our objectworld definition, backing up the pragmatic justifications of chapters 3 and 4.

At this point we should reiterate that the division between our three kinds of system can be indistinct, and often depends as much on the uses a teacher encourages students to put them to as on inherent design features. Likewise, the learning experiences we have just described blur at the edges. What we have been attempting is to indicate where each kind of system might be most useful. In a similar vein, Hammond and Trapp (1992) propose a methodology which teachers and designers can use to help them match educational tool to learning situation. Their scheme entails filling a matrix which pits software approach against learning scenario with values deriving from the three kinds of analysis shown in figure 5.18 below. Hammond and Trapp give a list of heuristics to assist in the interpretation of the matrix.

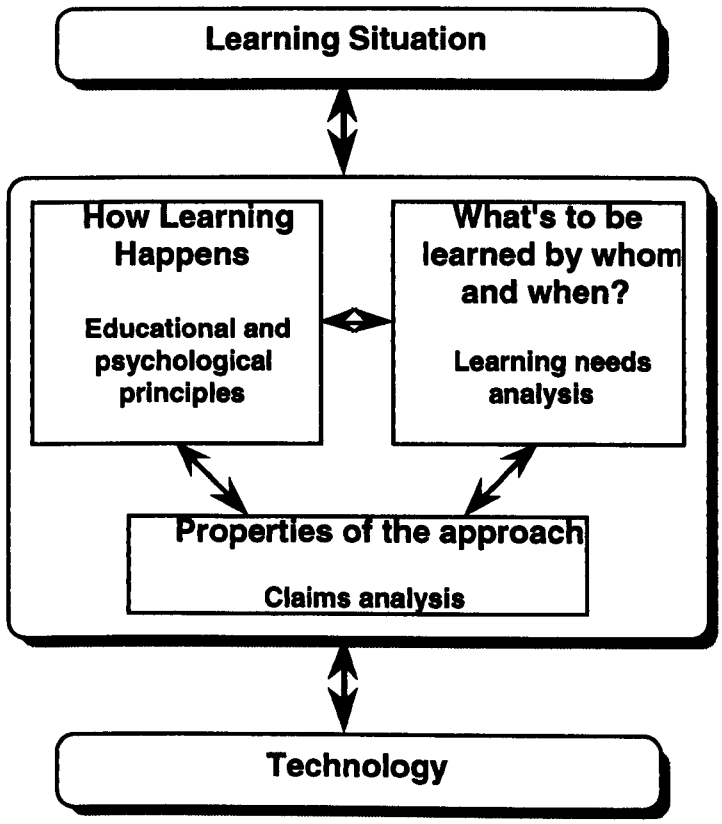


Figure 5.18 Bridging between learning situation and technological artifact (Hammond and Trapp, 1992)

The analysis of Learning Needs (the right hand sub-box) is beyond the scope of this thesis. Indeed, it depends on external factors such as curriculum

goals or course design, which are the concern of teachers. However, the critique in this chapter makes a contribution to knowledge of the Properties of the Approach (the bottom sub-box). In the next chapter, we give some consideration to the educational and psychological principles underlying objectworlds (the left hand sub-box).

6 Transitional Objects and Syntonic Commands

6.1 Overview

Transitional objects and their associated syntonic commands are obviously very important to the view of objectworlds we have put forward in this thesis. However, actually defining them in concrete terms is not really feasible; it would be rather like trying to define poems or sculptures. Consequently the definition of an objectworld given in chapter 1 does not mention them explicitly, but makes only general requirements.

Nevertheless, it would be unsatisfactory to leave our comments on transitional objects and syntonic commands in this state. One of the aims of this thesis is to assist those who would like to build their own objectworlds. Accordingly, we have been quite specific about their essential characteristics. If we now have to abandon prescriptive definitions, then we must provide a substitute of some kind.

In fact, there are important statements that can be made about transitional objects and syntonic commands but these statements are descriptive in nature. In this chapter we bring together a collection of such comments with the intention of providing a way of thinking about them that will be useful to future objectworld designers. The chapter sets the scene for the actual work of transitional object design.

We begin with an analysis of the statements Papert has made over the last two decades or so, mainly in the context of his work with children using Logo and Turtle Geometry. For Papert the Turtle is a transitional object - it connects both to everyday experience and to formal mathematical ideas.

Next we will examine the theories of the clinical psychologist Donald Winnicott, who used Freudian psychoanalysis to help with the treatment of emotionally disturbed children. The part of his work which concerns us began with the publication of his paper *Transitional Objects and Transitional Phenomena* (Winnicott, 1951) and continued into the 1970s. For Winnicott, a transitional object is some graspable thing that very young infants fix on at a critical, pre-perceptual stage in their development. For Winnicott this object, perhaps a toy or the corner of a blanket, is essential to the child's structuring of the world into 'me' and 'not me'.

Finally, we survey the contribution of the educationalist Robin Hodgkin, whose 1985 book *Playing and Exploring* (Hodgkin, 1985) seeks to unify the work of Winnicott and Papert in a theory of the ways in which transitional objects can encourage children to enjoy and be unafraid of discovery learning.

6.2 *Papert's Concept of Computer-based Transitional Objects*

Soon after the construction of Logo and the first Turtles, Seymour Papert began to build psychological foundations for the novel approach to learning that the new systems offered. From the beginning his objective was to offer children a more concrete programme of school mathematics, with a shift in emphasis away from abstract formulae, which many learners find unmotivating, towards meaningful creative activities. The titles of some of his papers from this period reflect his intent - "Teaching Children to be Mathematicians Versus Teaching Children Mathematics" (Papert, 1972), and "Teaching Children Thinking" (Papert, 1970) for example. Papert viewed the Turtle as a vehicle for these new activities.

Throughout the 1970s Papert and other researchers studied children using Logo with a variety of objects. We described some of these in chapter 1: conventional Turtles, velocity and acceleration Dynaturtles, sprites and so on. As a mathematician, Papert's instinct was to try to generalise this new concept of manipulable computational objects, but as a former colleague of the eminent psychologist Jean Piaget, he wished also to link them to psychologically plausible ideas about the learning mechanisms of children. Papert's unification of these twin aims grew from memories of a powerful learning experience he had as a child. In the preface to *Mindstorms* (Papert, 1980) he describes his childhood fascination with mechanical gears and tells how later on gears became an important aid to his acquisition of the concepts of arithmetic and algebra:

> "Gears, serving as models, carried many otherwise abstract ideas into my head. I clearly remember two examples from school math. I saw multiplication tables as gears, and my first brush with equations in two variables (e.g. $3x + 4y = 10$) immediately evoked the differential." (Papert, 1980 p vi)

This last comparison is striking. The differential is a system of gears which allows the two wheels of a driven axle to rotate at different rates (the two terms on the left hand side of the equation) even though they are both connected to the same propeller shaft turning at a constant speed (the right hand side, 10). It is the differential which allows an automobile to travel around curves while still driving both wheels. Reflecting on these thoughts Papert was able to make a synthesis:

> "Piaget's work gave me a new framework for looking at the gears of my childhood. The gear can also be used to illustrate many powerful "advanced" mathematical ideas, such as groups or relative motion. But it does more than this. As well as connecting with the formal knowledge of mathematics, it also connects with the "body knowledge", the sensorimotor schemata of a child. You can be the gear, you can understand how it turns by projecting yourself into its place and turning with it. It is this double relationship - both abstract and sensory - that gives the gear the power to carry powerful mathematics into the mind. In a terminology I shall develop in later chapters, the gear acts here as a *transitional object.*" (Papert, 1980 p viii)

The transitional object is not quite a tool (Papert does not suggest he used real gears to carry out an actual calculation) nor is it what a mathematician or philosopher would call a symbol, as it does not stand for a particular value or proposition but instead points at an idea. The object may take on one or other of these roles (tool or symbol) later, but they are not part of its initial purpose. It is associated with a cluster of meanings in the learner's mind; to use a term from clinical psychology, it has become *cathected.* It is Papert's contention that such transitional objects, inspired by real entities, are a natural and common feature of human learning. For some of us gears might do the job, for others a balance beam or the hands of a clock could be the device. Papert's insight was that the computer offers a *new medium* in which educators can build transitional objects for learners:

> "What the gears cannot do the computer might. The computer is the Proteus of machines. Its essence is its universality, its power to simulate. Because it can take on a thousand forms and can serve a thousand functions, it can appeal to a thousand tastes." (Papert, 1980 p viii)

From this point of view the Turtle is the first example of a new breed - *computer-based* transitional objects. In another paper Papert discusses other examples, such as the Sprites we touched on in chapter 1:

> "A sprite is something you can touch; it's there, it's an object. It has a colour and movement. You can give it a shape and you can change its shape. You can do something to it and it will change and it will act. So, in some ways, it's a little like these things we work with in the real world, and in some ways it's like those abstract ones. This ability to create transitional objects gives us a way of closing the gap between intuitive and formal learning." (Papert, 1987a p88)

This quote also indicates more clearly the purpose of transitional objects. Papert sees them assisting young learners to make the jump from concrete to formal ways of thinking. To help them move from manipulating the things in front of them, to using their imagination to explore the *space of possibilities* the same things represent.

> "For Piaget, what makes up the formal stage is really symbol manipulations. Propositions that refer to propositions. Thinking that refers not to a concrete reality but to a representation of reality and to all the possible situations that could arise under given real constraints." (Papert, 1987a p93)

This is a key feature of transitional objects in Papert's scheme, but in fact he has even grander plans for them. There are hints of this in *Mindstorms* but it is in subsequent papers that he makes clear his aim: he wants to *rehabilitate* concrete or intuitive ways of knowing from their Piagetian position of inferiority with respect to formal, analytic cognitive processes:

> "Where concrete approaches to learning have been recognized at all, it has most often been as inferior ways of knowing, the kinds of knowing adopted by necessity by those who have not yet mastered the canonical style. Thus Jean Piaget recognizes in young children ways of thinking that do not conform to the canon but that are too coherent and efficacious to be branded simply as "wrong." He casts children's concrete thinking as a stage in a progression to a formal style." (Turkle and Papert, 1990)

Computer-based transitional objects are the means to this revaluation of the concrete. So it is not only that playing with the gears could help Papert, or anyone else, come to appreciate algebraic equations but also that they *feed new insights* into the topic:

> "[Piaget] talks almost entirely about cognitive aspects of assimilation. But there is also an affective component. Assimilating equations to gears certainly is a powerful way to bring old knowledge to bear on a new object." (Papert, 1980 p vii)

The possibilities for getting our hands on many of the concepts of mathematics and science are quite restrictive. The learner first has to serve a lengthy apprenticeship working and practising with the written formalisms developed for the topic. Papert gives examples of the distortions this situation has lead to in education. For example:

> "In physics, dynamics is traditionally taught after statics, even though this is obviously perverse. In the history of physics, it is clear that dynamics provides the fundamental driving force, the fundamental ideas about how things move ...There are many obvious reasons for this... the only time you formalize it satisfactorily is when you get into calculus - and to get into calculus you have to take this long complicated path..." (Papert, 1987a p86)

So, statics is taught first because it requires less sophisticated mathematical techniques, not because dynamics is based on it or because it is most important. In the opinion of Papert, computer-based transitional objects can help to change this situation:

> "One might say that the formal stage arrived so late precisely because there were no computers. Take the one aspect of manipulating symbols. You can readily manipulate blocks, or the technology of wood, but until now, you could only manipulate symbols in your head, or with the very abstracted means of pencil and paper. We didn't have any good way of externalizing the manipulation of symbols (and still don't apart from the computer), and certainly no way that's accessible to very young children." (Papert, 1987a p93)

This theme - stressing that concrete ways of learning scientific concepts can in the age of the computer be as powerful as the formal, propositional approach which deters so many children - has been a connecting strand throughout much of Papert's work. As a recent paper puts it:

> "...the computer has emerged as an important actor in the revaluation of the concrete, a privileged medium for the growth of alternative voices in dealing with the world of formal systems. The conventional route into formal systems, through the manipulation of abstract symbols, closes doors that the computer can open." (Turkle and Papert, 1990)

But there is an obvious weakness in Papert's argument. He fails to stress that another important ingredient has to be present for a simulated computational object to become transitional. The *manipulating* of objects that we have been referring to is not so straightforward as it may seem. We want to stress, more explicitly than Papert has, that the ways we can manipulate the object are of equal importance to the object itself. Without `forward` and `back`, `right` and `left`, the Turtle would be of far less educational utility than it is. The Turtle, however, is something of a special case. Once one has

constructed an object with only position and heading as its properties, then the appropriate commands are fairly obvious. But as the attributes and behaviours of the object become more complex things get more difficult. In the case of Massobs, as we saw in chapter 2 (sections 2.2.2 and 2.8) the commands are harder to design, and there are more choices. We will consider this situation in a little more detail in the next section.

6.3 *Papert's Concept of Syntonic Commands*

In chapters 1 and 2 we introduced the concept of syntonic commands. As we pointed out in section 2.2.3, our characterisation is a refinement of ideas Papert put forward in *Mindstorms* (Papert, 1980), in that Massob boost commands are a little more demanding than the Turtle commands he discussed. We now wish to examine Papert's ideas more closely.

In *Mindstorms* he points out that it is possible to describe, in very simple terms, a method for drawing a circle: "move forward a little, turn a little, repeat the process." This description is easily translated into a Turtle Geometry procedure:

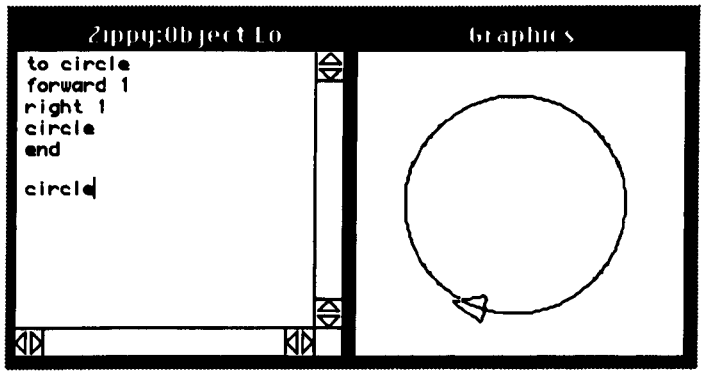


Figure 6.1 The Turtle Geometry circle.

Papert characterises this in the following way:

> "The Turtle incident illustrates *syntonic learning*. This term is borrowed from clinical psychology and can be contrasted to the dissociated learning already discussed. Sometimes the term is used with qualifiers that refer to kinds of syntonicity. For example, the Turtle circle is *body syntonic* in that the circle is firmly related to children's sense and knowledge about their bodies. Or it is *ego syntonic* in that it is coherent with children's sense of themselves as people with intentions, goals, desires, likes and dislikes... Turtle geometry is learnable because it is syntonic. And it is an aid to learning because it encourages the conscious, deliberate use of problem-solving and mathetic strategies." (Papert, 1980 p 63-64)

A critical flaw in Papert's position is that he fails to emphasise that the commands **`forward`** and **`back`**, **`right`** and **`left`** are as responsible for this syntonicity as the Turtle itself. The Turtle's state is its position and its heading. If children were left to manipulate it with the direct commands

`setposition` and `setheading`, there would not be the same opportunity for syntonic learning. Papert is no doubt aware of this, but he does not explicitly identify the *separate contributions* which attributes and commands make to the potential of transitional objects. We believe it is a point of key importance for designers of objectworlds.

Most sprite implementations are actually flawed in this respect; they lack reasonable syntonic commands. The state of a sprite is comprised of its position, heading and speed, and the only commands that are usually supplied are `setposition`, `setheading` and `setspeed`. These require an appreciation of vectors and coordinate systems for their use, mathematical ideas which are beyond most young children. Papert relates that first and second graders using sprite Logo at the Lamplighter school in Texas used sprites statically, simply editing their shape and colour because there were no appropriate syntonic commands (Papert, 1987a p82-83). Of course, as we outlined in section 2.8, it would be possible to implement suitable commands in Logo, perhaps similar to the Massob boost commands, and Lawler describes a syntonic scheme for sprites in his Beach microworld (Lawler, 1982).

One suggestion Papert does make is that educators should critically examine the subjects they wish children to learn and find opportunities for the use of transitional objects:

> "But if we can find an honest place for scientific thinking in activities that the child feels are important and personal, we shall open the doors to a more coherent, syntonic pattern of learning." (Papert, 1980 p 97)

He also points out that syntonic learning, and by implication syntonic commands, can draw on different *kinds* of experience:

> "One of the most widespread representations of the idea of angle in the lives of contemporary Americans is in navigation. Many millions navigate boats or airplanes or read maps. For most there is a total dissociation between these *live* activities and the *dead* school math. We have stressed that using the Turtle as a metaphorical carrier for the idea of angle connects it firmly to body geometry. We have called this body syntonicity. Here we see a *cultural syntonicity*: the Turtle connects the idea of angle to navigation, activity firmly and positively rooted in the extra-school culture of many children. And as computers continue to

spread into the world, the cultural syntonicity of Turtle geometry will become more and more powerful." (Papert, 1980 p 68)

These rather vague comments do not constitute a theory of transitional object design. However, they do indicate some of the directions educators should look in. It is important to realise that syntonic learning is not a new phenomenon, uniquely provided by computers and computer based transitional objects, but that it is a fundamental feature of human learning, given new form and possibilities by these devices:

"...it sounds as though ego-syntonic mathematics was recently invented. This is certainly not the case and, indeed, would contradict the point made repeatedly in this essay that the mathematics of the mathematician is profoundly personal. It is also not the case that we have invented ego-syntonic mathematics for children. We have merely given children a way to reappropriate what was already theirs." (Papert, 1980 p 206)

6.4 *Synthesis*

We believe that the character of the object and the means provided to manipulate its state are of equal importance in an objectworld. In our view, Papert fails to make a proper separation between the two components. Furthermore, in view of the practical findings of chapter 3 and the curriculum survey in chapter 4, we have to emphasise that we see a wider role for objectworlds than simply as an accelerator for children who are moving from concrete to formal operations in some new topic. Papert places this role above all others. We see the gaining of new experience and the exercising of skills as being of at least equal importance.

We also take issue with Papert on the central importance of using computers to rehabilitate concrete approaches to scientific topics. Certainly we believe this *might* be possible, but is it desirable? We feel it is risky to base much of the value of objectworlds on a radical new approach with an uncertain outcome.

In the view of some educators, many of Papert's claims are weakened by a lack of empirical support. It is true that Papert has provided little evidence, of the traditional kind, for the efficacy of his systems. Most of his papers simply report anecdotal details of student learning experiences. Papert's defence to this criticism is that he has always distrusted traditional methods of measurement. This is illustrated by a quotation from one of his earliest papers (a quotation we have already given in chapter 1 but which it is appropriate to use again):

> "This paper is dedicated to the hope that someone with power to act will one day see that contemporary research on education is like the following experiment by a nineteenth century engineer who worked to demonstrate that engines were better than horses. This he did by hitching a 1/8 HP motor in parallel with his team of four strong stallions. After a year of statistical research he announced a significant difference. However, it was generally thought that there was a Hawthorne effect on the horses... the purring of the motor made them pull harder" (Papert, 1970).

Several years later he showed again that he has almost instinctive doubts about what psychological testing can tell us:

> "...I find myself frequently reminded of several aspects of my encounter with the differential gear. First, I remember that no one told me to learn about

> differential gears. Second, I remember that there was *feeling, love,* as well as understanding in my relationship with gears. Third, I remember that my first encounter with them was in my second year. If any "scientific" educational psychologist had tried to "measure" the effects of this encounter, he would probably have failed. It had profound consequences but, I conjecture, only many years later. A "pre- and post-" test at age two would have missed them." (Papert, 1980 p viii)

More recently, Papert has worked his doubts into a more general critique in which he claims that some researchers have fallen prey to what he calls *technocentricity*. For example, several studies in the 1980s set out to measure the "efficacy" of Logo (e.g. Pea, Hawkins, and Sheingold, 1983; Clements and Gullo, 1984). The question asked is "What does Logo do to children?" The methodology used, the treatment methodology, has two components. First, Logo is introduced to a class, *keeping everything else constant.* Second, a *single particular thinking skill* is chosen, such as planning skills. The results of the Logo class in tests for this skill are compared against a control class.

Papert argues that both components of this methodology (applied to education) are flawed. The first is ruled out because it would be

> "a self-defeating parody of scientism to suppose that one could keep everything else, including the culture, constant while adding a serious computer presence to a learning environment. If the role of the computer is so slight that the rest can be kept constant, it will also be too slight for much to come of it." (Papert 1987b)

As for the second component, he argues that thinking skills cannot be singled out to any purpose and that a lower score in a particular test does not tell the story of a student's overall proficiency.

According to Papert, these kinds of experiment are examples of technocentrism. They ask a question that ignores the cultural dimension:

> "If I built a house out of wood and it fell down, would this show that wood does not produce good houses? Do hammers and saws produce good furniture? These betray themselves as technocentric questions by ignoring people and the elements only people can introduce: skill, design, aesthetics." (Papert 1987b)

In place of these technocentric experiments and their various flaws Papert advocates something quite different: a new genre of appraisal which he calls *computer criticism*:

> "I am proposing a genre of writing one could call 'computer criticism' by analogy with such disciplines as literary criticism and social criticism. The name does not imply that such writing would condemn computers any more than literary criticism condemns literature or social criticism condemns society. The purpose of computer criticism is not to condemn but to understand, to explicate, to put in perspective." (Papert, 1987b)

Unfortunately, before computer criticism can flourish, the culture holding it has to become fluent with the medium. Papert's proposal implies that designers of educational computing systems need to be computer literate to a much greater degree than is common at present, so that a popular discourse along the lines he envisages seems a long way off. However, Papert and his colleague Sherry Turkle have started the ball rolling with a couple of papers: *Computer Criticism versus Technocentric Thinking* (Papert, 1987b), which we have already referenced, and *Epistemological Pluralism: Styles and Voices Within the Computer Culture* (Turkle and Papert, 1990).

Of course, Papert's discussion of technocentricity will not satisfy everybody. It does not require excessive cynicism to wonder just how many dubious educational philosophies could be advocated if we were to exempt them from scientific examination and his proposed genre of computer criticism has yet to flourish. Furthermore, his criticism of the treatment methodology is really saying that it is simply very difficult, not impossible, to do good testing of computer assisted learning. However, from the point of view we have developed in this thesis it is as if Papert is tilting at windmills: most of the experiments he is so critical of (there are many more examples, for instance Burns and Hagerman, 1989; Cathcart, 1990; Swan, 1991) are simply investigations of Logo *programming* rather than enquiries into the properties of transitional objects or objectworlds.

So, in the present context, the important questions are "What can we do with objectworlds?" and "Can meaningful things be said about the nature of transitional objects?" In chapters 3 and 4 of this thesis we made a start on the first question (as it relates to Gravitas). In the remainder of this chapter we consider the second.

6.5 *Winnicott and Transitional Objects*

Seymour Papert was not the first researcher to write about transitional objects. In fact, the paediatrician and psychoanalyst Donald Winnicott introduced the concept in 1951, and he was followed by others, such as the psychotherapist J. C. Solomon (1962). It is difficult to know whether Papert was directly influenced by their work when he wrote *Mindstorms* in the late 1970s. So far as we can ascertain he does not make any reference to Winnicott until 1990 (Turkle and Papert, 1990). Furthermore, their areas of concern are somewhat different: Winnicott's interest is in the world of the very young human infant, in particular the development of awareness, while Papert focuses on human learning. Nevertheless, there are some striking similarities between the two concepts, and we feel that the psychoanalyst's approach may provide some useful insights to the educational exploitation of transitional objects.

To understand Winnicott's concept of transitional objects we first need to describe some of the psychological assumptions from which his ideas spring. One of the starting points for Winnicott is an expanded view of human nature. It is generally accepted by psychoanalysts that a description of human nature purely in terms of interpersonal relationships is not adequate. Any discussion of normal or pathological behaviour needs to recognise the existence not just of an *external reality* and an individual's interactions with it, but also an *inner reality* - "an inner world that can be rich or poor and can be at peace or in a state of war." (Winnicott, 1971 p2)

This was received opinion at the time of Winnicott's writing. However, he took things a stage further by urging an appreciation of

> "the third part of the life of a human being, a part that we cannot ignore... an intermediate area of *experiencing*, to which inner reality and external life both contribute." (Winnicott, 1971 p2)

When Winniccott put his thesis forward it was considered novel, and subsequently it has had great influence (see for instance Eichenbaum and Orbach, 1982 p112). In some ways, it is easier now for us to accept his idea. We can draw on computational metaphors, for example by comparing the "intermediate area of experiencing" to the interfaces that sit between a central processor and the various peripherals and sensors which might be connected

to it. But it is still a problematic area of human nature to discuss, especially since the vocabulary of psychoanalysis is still adapting to it. Winnicott's fundamental point is that this faculty, like any other, requires stimulation and nurturing for its proper development:

> "At the theoretical beginning a baby can be said to live in a subjective or conceptual world. The change from the primary state to one in which objective perception is possible is not only a matter of inherent or inherited growth process; it needs in addition an environmental minimum. It belongs to the whole vast theme of the individual travelling from dependence towards independence." (Winnicott, 1971 p151)

The provision of this "environmental minimum" is the responsibility of the mother or, more correctly, the adult in the maternal relationship. The mother places objects into the child's vicinity, in the beginning her breast (or a feeding bottle), but then clothing, bedding, and her limbs. In the natural sequence of events the faculty of experiencing is ready to be born:

> "at some theoretical point early in the development of every individual an infant in a certain setting provided by the mother is capable of conceiving of the idea of something that would meet the growing need that arises out of instinctual tension...There is an overlap between what the mother supplies and what the child might conceive of." (Winnicott, 1971 p12)

Into this environment comes what Winnicott calls a transitional object: an external catalyst for the beginning of awareness and the child's first structuring of the world into 'me' and 'not-me'. If we study a particular child in detail:

> "there may emerge some thing or some phenomenon - perhaps a bundle of wool or the corner of a blanket or eiderdown, or a word or tune, or a mannerism - that becomes vitally important to the infant for use at the time of going to sleep, and is a device against anxiety, particularly anxiety of the depressive type" (Winnicott, 1971 p4)

We must remember that the infant is still supposed to exist in a subjective world, so we cannot talk about the child *perceiving* the transitional object, perception being an active process which relies on prior knowledge. Winnicott explains this in a way which adds to the psychoanalytic meaning of the word 'create':

> "I should like to put in a reminder that the essential feature in the concept of transitional objects and phenomena (according to my presentation of the subject) is *the paradox and the acceptance of the paradox*: the baby creates the object, but the object was waiting to be created and to become a cathected object." (Winnicott, 1971 p89)

If we compare what we have seen so far of Winnicott's theories with Papert's concept of computer-based transitional objects two things immediately stand out. The first feature of note is that in both cases transitional objects are aimed at episodes of large scale mental change: the development of apperception on the one hand; the move from concrete to formal thinking on the other. Next, we see a similarity in the roles of mother and teacher: both are responsible for providing an environment which contains objects the individual can appropriate, although we would usually say the learner 'discovers' the Turtle and its behaviour.

Returning to Winnicott, we find he has much to say about the character of transitional objects. For instance, he stresses that the relationship between the child and the object is affectionate. This property is reminiscent of Papert's description (in the foreword to *Mindstorms*) of falling in love with the gears he played with as a child. Winnicott also emphasises that the object has to be something that the infant has the capacity to create (in his new sense of the word). This injunction echoes Papert's suggestion that we should equip transitional objects with methods of manipulation (the syntonic commands of section 6.3) that the learner already understands. Another observation of Winnicott's also has similarities with the picture we have presented in this thesis: he insists that the object must never change (unless it is changed by the infant), if it changes unpredictably then the foundations of the infant's perceptions are shaken. For similar reasons the definition of an objectworld presented in chapter 1 made the demand that the object be continuously visible.

We mentioned in section 6.2 that a computer based transitional object is not a symbol in the usual sense. Winnicott comes to a similar conclusion:

> "It is true that the piece of blanket (or whatever it is) is symbolical of some part- object, such as the breast. Nevertheless, the point of it is not its symbolic value so much as its actuality... When symbolism is employed the infant is already clearly distinguishing between fantasy and fact, between inner objects

and external objects, between primary creativity and perception... It would be possible to understand the transitional object while not fully understanding the nature of symbolism." (Winnicott, 1971 p6)

However, it is interesting to read Winnicott's views on what happens to a transitional object after it has performed its role as a catalyst for mental change:

"Its fate is to be gradually allowed to be decathected, so that in the course of years it becomes not so much forgotten as relegated to limbo. By this I mean that in health the transitional object does not 'go inside' nor does the feeling about it necessarily undergo repression. It is not forgotten and it is not mourned. It loses meaning, and this is because the transitional phenomena have become diffused, have become spread out over the whole intermediate territory between 'inner psychic reality' and 'the external world as perceived by two persons in common', that is to say, over the whole cultural field." (Winnicott, 1971 p5)

For Papert the object that has ceased to be transitional may still be *useful* in at least two senses. First, one can still do things with the object long after one has learnt the important concepts for which it was a vehicle. For instance one can go on using a Turtle to do drawings, and Massobs can be used to carry out physical experiments. In this way the objects become more like tools. Secondly, as we explained in section 6.2, the transitional object is permanently a means of doing the formal in a concrete way: we could, for example, use a Turtle to investigate trigonometric theorems.

Winnicott, however, sees this move towards *use* of the object as a problematic issue. The difficulties are caused by the shift in the role of the object from its initial purpose as the instigator of a new mode of thought, to becoming a channel for the expression of thought. As Winnicott puts it, in a phrase which reminds us that his interest in the field derives from his therapeutic work:

"In the sequence one can say that first there is object relating, then in the end there is object use; in between, however, is the most difficult thing, perhaps, in human development; or the most irksome of all the failures that come for mending." (Winnicott, 1971 p89)

He makes an important distinction between *relating* to an object and using it. It is Winnicott's opinion that

> "...relating [to an object] can be described in terms of the individual subject, and that usage cannot be described except in terms of the acceptance of the object's independent existence, its property of having been there all the time." (Winnicott, 1971 p88)

Relating and using are so different that the child must break down the charge of mental energy associated with a transitional object before the new ability to use objects can be applied to it:

> "This sequence can be observed: (1) Subject *relates* to object. (2) Object is in process of being found instead of placed by subject in the world. (3) Subject *destroys* object. (4) Object survives destruction. (5) Subject can *use* object." (Winnicott, 1971 p94)

And this ability to use things is a separate talent we all have to learn, which also needs nurturing, in Winnicott's famous phrase, by a "good-enough mother":

> "To use the object the subject must have developed a *capacity* to use objects... This capacity cannot be said to be inborn, nor can its development in an individual be taken for granted. The development of a capacity to use an object is another example of the maturational process as something that depends on a facilitating environment." (Winnicott, 1971 p89)

We have gone as far as we should in our examination of Winnicott's theories. The overall purpose of his research was to give insights to the condition of emotionally disturbed children who came to him for therapy. He believed that many psychopathological problems could be traced back to an unsatisfactory relationship with transitional objects in infancy. Accordingly, his enquiry into objects becoming tools was less strong, whereas in the context of education this process is very important. However, there is another researcher who has taken up this point, and it is his work we will consider in the next section.

6.6 *Hodgkin: Transitional Objects and Play*

During the 1950s Robin Hodgkin was headmaster of Abbotsholme school, where his background as an enthusiastic climber and mountaineer led him to give outdoor activities an important place in the curriculum (Hodgkin, 1980). He then moved to Oxford university, where he was a professor of education until 1979. A common theme in Hodgkin's academic work was the search for a firm psychological basis to his belief in the educational value of playful and practical pursuits. In transitional objects he found a touchstone and in *Playing and Exploring* (Hodgkin, 1985), a book written in his retirement, he sought to amalgamate the views of Winnicott and Papert with his own.

As may be guessed from the title of his book, Hodgkin's special interest is play and its relationship to discovery learning. Accordingly, he focuses on objects which children can first encounter as *toys*. He stresses that profound educational developments take place as children convert the objects in their environment from *toys* to *tools*, or from *toys* to *symbols*. It is possible to make "distinctions in the understanding of all cultural objects, including words and pictures: that they can be used with only slight intent (toys), with precise intent (tools) or to cope with ambiguities and with multiple levels of understanding (symbols)." (Hodgkin, 1985 p40)

This leads to a refinement of the picture of transitionality we have presented so far. If we re-phrase everything in Hodgkin's terms then Winnicott's transitional objects become tools as they "become decathected". Papert's computer-based transitional objects change from toys into tools as they carry their powerful ideas into a child's mind. The move from tool to symbol, which is only briefly touched on by Winnicott and which Papert downplays, is really Hodgkin's territory:

> "Seen from the point of view of 'things for use' transitional objects can be regarded as the juvenile source from which flows all the practical gear of a technical world; but from another perspective - 'things for meaning' - they are the beginnings of all our imaginative and intuitive dreaming, of poetry and of religion." (Hodgkin, 1985 p42)

So where does Hodgkin see transitional objects fitting into education? A metaphor for meaningful discovery learning he frequently refers to is "frontier experience" which can mean an explorer seeking a new route up a mountain,

or a child working at the outer limit of her knowledge. But this kind of activity, to which he attaches so much importance, is not always pleasant for everyone. Some learners are frightened if they are not given a set route through a topic. Hodgkin sees an important role for transitional objects :

> "Why is novelty, frontier experience, sometimes so alluring - as with [Papert's] gear wheels - yet sometimes so repellent? Bruner has given us the term 'pre-emptive metaphor' to describe how it is that at times of sharp developmental transition a boy or girl may be blocked from learning by a cluster of ideas which link associatively with one central locus of fear (Bruner, 1968 chap. 7). It is useful to have a similar but positive terminology for those objects and experiences which *foreshadow success in exploring*... Such an opening is made available or entered into through the action of some transitional object becoming a symbol." (Hodgkin, 1985 p44)

What is it like, this process of toy becoming symbol? Consider a child coming to the Turtle for the first time. At the beginning play seems to have little purpose, simply producing squiggles on the screen. After a while, perhaps under the prompting of a teacher, the child learns to draw specific figures. With practice the child becomes more proficient: the Turtle (in this context) has become a drawing tool. Next the child returns to playful activities but is playing with a tool "which points to a cluster of intentions." (Hodgkin, 1985 p40). At some stage, and again with help from a teacher, this play might lead to a procedure which generates families of drawings, say a sequence of similar houses which increase in size. For Hodgkin, this is the genesis of symbol use and it is a key process for education to encourage.

> "How are these concepts - toys, tools and symbols - related? This question runs parallel to the crucial educational problem of how play, practice and creative discovery merge into each other within a heuristic field... Play, practice and exploration: we need to hold these together in a model which is coherent and which matches our common experience of teaching." (Hodgkin, 1985 p44-45)

Hodgkin emphasises that the processes of 'toy becoming tool' and 'toy becoming symbol' are *not* independent steps in a sequence:

> "In trying to think clearly about the relationship between play, practice and discovery. I found myself in a cul-de-sac. The reason was that I had imposed a mistaken sequence on the three kinds of artefact on which we act in these roles.

Toys, I thought, were the most primitive; tools must develop out of toys and then symbols must develop somehow out of tools. And yet that never seemed right... we seem to require a two-way oscillating and dialectical concept, more on the lines of Taoism's *yin* and *yang* and less like a linear progression." (Hodgkin, 1985 p51-52)

In *Playing and Exploring* he gives a diagram to help explain his point:

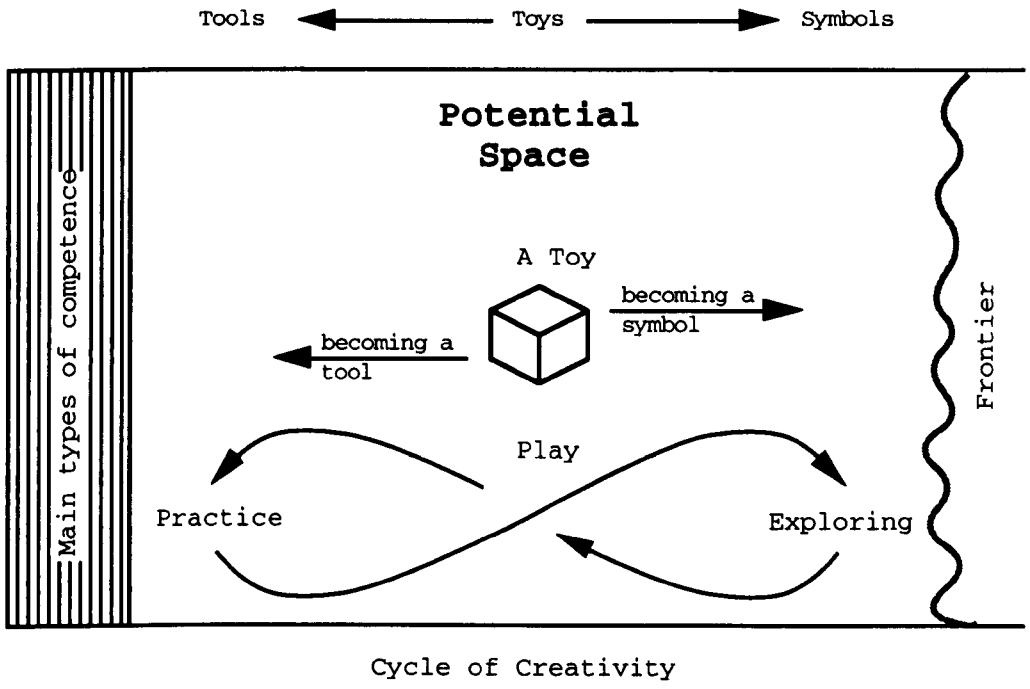


Figure 6.2 The cycle of creativity: "Things a person plays with can, with practice, become part of some skill and thus eventually assimilated into a general area of competence. However, a toy or 'playwith' can also be pushed out to the frontier and become what might be termed an 'explorewith'." (Hodgkin, 1985 p46)

Play is always the driving force for Hodgkin. Playing with transitional objects can create tools or symbols according to the learner's will:

"We can either move from [play] in the direction of increasing efficiency and control, as we consolidate our competence, or we can move towards uncertainty and challenge as we stretch our competence." (Hodgkin, 1985 p52)

He even offers a mild rebuke to Piaget for neglecting a dimension of play:

"[Piaget] has also been responsible for a one-sided view of play, which he sees as being essentially a kind of repetitive practice. Play, seen in these terms, as helping learners to assimilate new concepts or skills to existing mental patterns ("schemas") is not enough. Piaget failed to emphasize the complementary direction in which play is also useful: that which leads learners toward exploration and to the more stressful and challenging processes of *accommodating* unfamiliar experiences and concepts." (Hodgkin, 1985 p48)

We have completed our discussion of what Hodgkin has to say on the educational uses of transitional objects. However, he is a great pragmatist and his theoretical musings always lead to practical advice. In that spirit we finish this section off with some recommendations which he made in a very general tone but which seem to apply well to computer-based learning environments, and especially to objectworlds:

> "The following maxims summarize the links between all three phases of the creative cycle and good teaching.
>
> (i) Teachers should be able to initiate a range of activities which covers the play-practice-play-discovery cycle.
>
> (ii) The range needs to be wide enough to match and extend the different levels of competence of learners in a given group.
>
> (iii) The quality of a student's action is a good test for the beginning of an educational episode or of a plan of learning. Similarly, the quality of feedback resulting from it - what he or she learns from successes or failures - is the crucial test for judging its conclusion.
>
> (iv) The range needs to be sufficiently varied that, at any given level, allowance is made for some relearning and overlearning of skills, and for some analysis of error without producing boredom." (Hodgkin, 1985 p53)

In respect of point (i) it should be apparent that objectworlds like Turtle Geometry and Gravitas do allow a wide range of activities within the play-practice-play-discovery cycle. Point (ii) is answered, in the case of Gravitas, by the combination of the easy to use graphical interface and the powerful programming language, which facilitates longer and more complex projects. As for point (iii), continuously visible objects, with consistent behaviours make it easy for students to devise actions. They receive feedback to their actions in a particularly direct fashion: a Turtle moves where it was expected to, or perhaps not; Massobs orbit each other, or collide, or escape. Activities such as those mentioned in Hodgkin's point (iv) are evident in the transcripts of the Moon shots in chapter 3. The students repeatedly try to get the boosts right but in a different context each time, and within an overall goal which they are motivated to attain.

6.7 *Summary*

We have described Papert's position which contains two main assertions. If you have an apprehendable object which connects to rich concept areas then it can be used to revaluate the concrete ways of knowing in its related domain and it can assist the learner's transition from concrete to formal operations in that domain. We have outlined our three main qualifications of this position which are (i) We believe that the object and the commands by which it is manipulated are of equal importance in an objectworld. (ii) We do not confine the application of objectworlds to children's episodes of transition between concrete and formal thinking. (iii) We believe that it is only very difficult, rather than impossible, to carry out rigourous investigations into the effect of objectworlds on children's learning.

We have outlined Winnicott's concept, derived from his clinical observations, of a critically important "first possession" which catalyses the development of awareness in young infants and which marks the start of their structuring the world into "me" and "not me". We have shown that although Papert's and Winnicott's transitional objects are part of quite different phenomena, there are striking resonances between them. In particular, they both emphasise that affection and consistency are essential features of the relationship between child and object. Winnicott and Papert also agree that learners' use of transitional objects prefigures (and is an important stage on the way to) the use of symbols.

Finally, we have shown how Hodgkin has combined the views of Papert and Winnicott to produce an educational theory of toys. In Hodgkin's view, a child playing with toys that are transitional objects can develop them into *tools* or *symbols*. Tools have direct practical relevance as "the juvenile source from which flows all the practical gear of a technical world." (Hodgkin, 1985 p42) For Hodgkin, the same key that opens the door to the realm of symbols can open the way to meaningful discovery learning.

7 Contributions and Further Work

7.1 Introduction

In this thesis we have identified a distinct class of computer-based discovery learning environments, which bring together a simulated object (or objects) and a programming language. The name we have given these systems is *objectworlds*, a refinement of the existing term *microworld* which has been applied to a wider range of software. In the sections which follow we detail the three main contributions this thesis makes to knowledge about objectworlds, and we point out some directions for future research.

7.2 *Contribution 1: The Identification of the Objectworld Class*

An important aim of this thesis has been to pick out objectworlds from the spectrum of educational computing systems and to describe their essential features. To achieve this aim we have done three things. First, we have shown that the kinds of environment we are talking about have a traceable history. Second, we have developed more definite meanings for terms which have been used quite loosely. Third, we have contrasted objectworlds and the kinds of interactions they can support, with their close neighbours - simulations and modelling systems.

7.2.1 *History of Objectworlds*

We have illustrated the historical development of this class of environments with references to example systems including Turtle Geometry, Turtle Biology, Dynaturtles, Sprite Logo, Boxer and LEGO/Logo. Our survey shows how objectworlds arose from early research into the use of interactive computer programming as an educational tool. We have described the developments which have led from the original Turtle to the more sophisticated central objects, such as Boxer sprites, of the present day.

7.2.2 *Establishment of Definite Meanings for Vague Terms*

We have shown that the existing name 'microworld', which was originally applied to Turtle Geometry and related systems, has now become a catch-all term used for a wide range of discovery learning environments. We have constructed a precise definition for a narrower class of systems. This definition discriminates objectworld from non-objectworld and also serves as a basic set of requirements for future objectworld designers.

Two concepts which are highly pertinent to objectworlds are *transitional objects* and *syntonic commands*. However, discussions about them have in the past been somewhat scattered and vague. In chapter 6 of this thesis we have brought together the relevant opinions of Seymour Papert, a Logo pioneer, Donald Winnicott, a clinical psychologist, and Robin Hodgkin, an educationalist. Our intent has not been to carve these concepts on tablets of stone. Rather, we have sought to describe a way of thinking about them which will prove useful and stimulating to future designers of objectworlds.

7.2.3 *Objectworlds and other Discovery Learning Environments*

We have compared and contrasted objectworlds with their near neighbours among educational computing environments - simulations and modelling systems. Our survey shows that although the three kinds of program have much in common they afford very different learning experiences for their users. In particular they all use *models* to carry their message, but they differ in what the model is used to convey.

In an objectworld the model is used to give a concept rich behaviour to a continuously visible object, which the learner can manipulate very straightforwardly. In a simulation, the model is encased in an interface which makes it easy for the user to vary parameters and observe results. In a modelling system the workings of the model are laid bare and users can choose to alter parameters or details of the algorithms; they may also construct their own models.

Our discussion of the three kinds of system has made it possible for future debate to be clearer about the role each can play in education. In this context we have speculated that objectworlds, simulations and modelling systems form three steps in a progression for learners. Objectworlds allow students to gain concrete experience in formal domains. Simulations allow them to develop and stretch their formal knowledge. Finally, modelling systems give learners the chance to carry out activities similar to those of a working scientist.

7.3 *Contribution 2: A New Example of the Class - Gravitas*

We have described a new objectworld called Gravitas, which provides learners with new simulated entities called *Massobs*. These behave like massive bodies moving through space obeying Newtonian laws of gravitation. All aspects of Massobs - position, velocity, mass and radius - may be controlled by commands or procedures typed into the Logo interpreter which runs concurrently with the system. Gravitas extends the concept of objectworlds in three main ways, which we describe below.

7.3.1 *A New Transitional Object*

With the creation of Massobs we have introduced a new transitional *object*. These are straightforward for students to work with and yet they connect to powerful physics concepts. Like Turtles, Massobs have a small set of *syntonic* commands. We use this term to describe commands which are particularly easy for young learners to comprehend because they connect to things children can be expected to know. In the case of Massobs, the four syntonic boost commands are simple to understand because they relate to sensorimotor knowledge about being pushed, and they require no understanding of coordinate systems or vectors for their use.

7.3.2 *Complex Behaviour*

Massobs exhibit a more complex dynamic behaviour than the central objects of previous systems such as Turtles and Sprites: they move continuously *and* their trajectories are affected by their gravitational interactions with each other. An underlying mathematical mechanism computes the forces acting between all the Massobs present in the system and modifies their velocities accordingly

7.3.3 *A Graphical Interface*

We have duplicated the functionality of *all* the Massob commands in a graphical interface which makes it easier for learners to create and manipulate Massobs in their two dimensional space. In fact, Massobs may be positioned and given a velocity, a mass and a radius by straightforward *direct manipulation* with the mouse. The graphical interface was installed in Gravitas to 'lower the threshold' which newcomers to the system must overcome before they can begin useful work.

7.4 *Contribution 3: An Initial Study of Learning Activities Supported by Gravitas*

Gravitas is a *new* system which presents a new educational space to be explored, not just by learners but also by educators who might wish to exploit it for their own teaching goals. In this sense, Gravitas is in a similar position to Turtle Geometry in the early 1970s, when Papert and others sought to develop a range of meaningful activities for children to engage in. Accordingly, we have carried out preliminary studies to identify the nature of the activities that Gravitas can support. We have made four principal discoveries which we detail below.

7.4.1 *Surprises*

First, learners are very often *surprised* by the behaviour of the Massob systems they construct, even when these are quite simple. The orbital procession of the Moon, for instance, surprised all of our subjects yet it arose from a system of only two Massobs. In our studies, subjects have been highly motivated to resolve such surprises, a process which requires them to think deeply about the physics of gravitating bodies. We have shown that it is possible to break a surprise down into constituent episodes, each of which learners *are* capable of resolving, in such a way that they are led to an overall understanding.

7.4.2 *Programming Gravitas*

Second, we have found that quite simple programs, well within the scope of novices, can generate interesting projects, such as the navigation of a rocket from the Earth to the Moon and back. There are no unusual limitations on the Logo interpreter attached to Gravitas and users are free to build arbitrarily complex programs. However, it was important to us that even a very basic level of programming could lead to meaningful work with Gravitas. It is significant in this context that some of the subjects who completed the Moon mission had no previous experience of writing programs.

7.4.2 *Interface Synergy*

Third, we have discovered a synergistic effect between the Logo programming interface and the graphical interface which allows learners to take on more complex tasks than would otherwise be the case. The Moon shot,

for example, is greatly facilitated by this synergy. The graphical interface encourages tactical investigations - does a particular boost have the required effect? - while programming supports a strategic approach - does that boost fit into the overall plan?

7.4.3 Gravitas and the Science National Curriculum

We have surveyed the Science National Curriculum and discovered several areas where Gravitas may be of direct application. In particular, Gravitas based activities seem to be appropriate for Statements of Attainment dealing with concepts such as Force, Momentum, Energy and Gravity, and with more general knowledge of astronomical bodies and satellites. These activities can be immediate, with students use Gravitas interactively, or procedural, with the learner exploiting Gravitas' programming interface to build Logo programs for some longer term goal.

7.5 *Further work - Theoretical Issues for Objectworlds*

This research has also raised a number of theoretical issues which require further investigation.

7.5.1 *Transitional Objects and Intuitive vs. Formal Knowledge*

Papert and others have claimed that transitional objects can help young learners make the leap from their intuitive, concrete knowledge about the way the world works to the formal and symbolic forms of knowledge which dominate science. Gravitas represents an attempt to give children formal objects - Newtonian masses - in a transitional form - Massobs, which the user can manipulate in concrete ways.

We have shown that Massobs can be useful in realisitic learning settings but Papert's assertions remain untested. The question is, can we discover whether the claims made for transitional objects are true? Can we measure any improvement in children's grasp of formal concepts, which is solely due to their interactions with transitional objects? Papert has cautioned against experiments of the usual kind, based on the treatment methodology, where one group is exposed to the new method and a second group is not, and results are compared. As we saw in section 6.4, he sees this as pointless *technocentrism*. Papert's solution is a new genre of writing which he calls *computer criticism*. Several references point the way, for instance: (Papert, 1987b; Harel and Papert, 1990; Turkle and Papert, 1990).

What is needed then, is a study in this spirit, which critically examines the development of children's understanding of a formal concept, like Conservation of Energy, as they use and play with Massobs. Chapter 4 shows how such a concept can be *demonstrated* in Gravitas, but the hard part, accurate observation and record, remains.

7.5.2 *Alternative Syntonic functions*

As we noted in section 2.8, the existing **boosts** are not the only form that syntonic commands for Massobs could have taken. At present they are suited to the kind of investigations we wanted learners to make for the studies in chapter 3, but it would be interesting to explore other ideas. For instance, we could try the scheme outlined in section 4.2.2, or Massobs could be given a new attribute called *boost heading* and syntonic commands analogous to the **left**

and `right` of the turtle. There would then be just a single `boost`. Such configurations would certainly suit investigations of single Massobs and the effect of forces on them, or situations where gravity is not a significant factor.

7.5.3 How Useful are Dual Interfaces?

In chapter three we reported on users carrying out a task which would be very difficult, if not impossible, to perform using a single interface. However, we have not had time to quantify the extent to which the dual interface accelerates progress. The effect would be fairly straightforward to investigate, for instance by setting the same task to students using different single interface versions of Gravitas.

7.5.4 Visual Programming Languages

It has often been claimed that learning to program carries general cognitive benefits. Some have even seen this as a purpose of transitional objects. For instance, Lawler, writing in the context of Turtle Geometry and sprites, states that:

> "People can become engaged with computational objects which they interpret as symbols for real objects. But they can only manipulate these objects by means of a computer language. Doing so engages them in the nitty gritty effort of learning a set of operations which transform the states of objects, and this gives them everyday experience with the surface details of a formal system whose deeper properties they can gradually come to appreciate." (Lawler, 1987 p15)

We have shown with Gravitas that it is possible to remove the need for people to use a programming language in the first instance, but we have also shown that there are interesting tasks which a graphical interface cannot tackle alone, and in these cases the user *must* program. However, programming languages, even Logo, still deter many learners with their fussy syntax and strict grammar. It would be interesting to take Gravitas one step further by replacing Logo with a visual programming language (Myers, 1986; Chang, 1987; Shu, 1986). That is, replace the typing in of Logo procedures with program construction by the graphical manipulation of iconic components.

7.6 *Further work - Practical Experiments with Gravitas*

Now that Gravitas has been built and we have found some learning activities it can support, the next logical step is to use the program with more children, in real educational settings. Gravitas is a robust system, not just a research prototype. It would work in school classrooms without breaking and therefore it would make sense to examine the use of Gravitas in real school science lessons.

7.6.1 *A Gravitas Physics Curriculum*

An exciting continuation of the work described here would be to create a physics curriculum based on Massobs in a manner analogous to the mathematics course represented by *Turtle Geometry* (Abelson and diSessa, 1980). We have shown in chapters 3 and 4 that Gravitas can support many different kinds of physics rich activities. Obviously, the construction of an entire curriculum would be a major task, and one best carried out with the co-operation of working teachers. However, we believe Gravitas has the potential to support such an effort, and the examples we have given show that Massobs can be vehicles for the key physical concepts of mass, force, momentum and energy.

7.6.2 *Constructing qualitative explanations for surprises*

A consideration related to the previous point concerns the way in which surprises, such as the orbital procession of the Earth, are resolved by puzzled learners. As we have said before, these surprises occur naturally as learners play with Massobs and they can be highly motivating. In our studies we led subjects to explain orbital procession to themselves by considering what happens over quarter turns - "three o'clock to six o'clock" and so on. Other surprises (and they are a common occurrence, especially when the boost commands are used) will require teachers to offer different qualitative stories to help their students achieve understanding.

7.6.3 *Connecting Gravitas to an Intelligent Tutoring System*

Several authors have pointed out weaknesses in the Discovery Learning approach. For instance, John Seely Brown (1985) writes: "critics of discovery learning point to its inherent inefficiency - students can spend a lot of time *floundering* - and to the possibility that some students will never make the

discoveries that we think they, as educated people, should make". Echoing this point is Wallace Feurzeig, one of the pioneers of Logo: "without the aid of a teacher, many children do not learn in a Logo microworld. They are not able to set their own goals, to find effective methods of thinking about problems, or to acquire the skills of exploration, conjecture and inference. Left to themselves, they thrash about" (Feurzeig, 1984).

In chapter 5 we mentioned Elsom-Cook's Guided Discovery Tutoring synthesis (Elsom-Cook, 1990), which attacks this problem by bringing together Discovery Learning Environments and Intelligent Tutors to provide a system in which students are free to explore a subject domain, while a software tutor provides guidance. Since objectworlds are a variety of Discovery Learning Environment, they could obviously take a place in this paradigm if a suitable tutor was developed.

It would be interesting to construct such a tutor for the topic of orbital transfers like the Moon shots reported in chapter three. To carry out these journeys users must form an overall plan and then experiment with the boosts required to initiate each stage of the journey. They also have to translate these findings into simple Logo programs. A tutor could provide advice in each of these contexts.

The intelligent tutoring system MYCROFT (Goldstein, 1975), which was built at MIT, tackled a similar problem. First of all it was intended to assist in the debugging of simple Logo programs, just the sort of program we have seen Gravitas users building. Second, the output of the programs which MYCROFT was designed to debug is mainly graphical. The same is true for Gravitas.

MYCROFT operates by mediating between the picture that actually *is* drawn, the program that the user writes to draw it, and a *description* of the target picture to be drawn (encoded in a simple declarative language). The system's resources for this mediation are a knowledge of plans and debugging strategies, and what Goldstein calls a *Cartesian annotator* that describes the performance of the user's program in geometric terms.

It seems feasible that the planning knowledge and debugging technique concepts of MYCROFT could be applied in an agent for Gravitas without serious difficulty. An equivalent for the Cartesian annotator, however, would seem to require a substantial amount of research. However, it might be feasible

if the domain of proficiency of the agent is restricted to *families* of situations, like the Moon shots, where two bodies are in stable orbit around each other and a third object is navigated between them.

7.7 *Concluding Notes*

A great deal more could be said about the possibilities for extending and experimenting with Gravitas, but it is time to call a halt. We have identified three major contributions of the research, and eight promising directions for further investigations.

We set out with the intention of building a system similar in scope to Turtle Geometry. Succeeding at this goal has left us with the task of exploring Gravitas' scope, a task we have only just begun with the studies of chapter three. We hope that others may come along to continue these explorations.

Another goal was to provide guidelines for educators who would like to construct systems of this type. In the end, Gravitas stands as simply another example. It would have been gratifying to discover a set of design principles but this will have to wait for the future. However, we believe that the definition of objectworlds is a solid beginning.

The most problematic issue attaching to this work is the question of transitional objects. Is it really possible to engineer on the computer objects which make concept acquisition easier for learners? In Massobs we have created another candidate but, as we have pointed out above, there is far more work to be done. However, we believe, along with Papert, that transitional objects may be one of the most profound contributions computers can make to education.

Appendix A - Dynamical Astronomy

A.1 The N-Body Problem

The gravitational behaviour of Massobs is generated by a mathematical mechanism which is part of a rich tradition. This tradition began with the invention of fast digital computers in the early 1950s and continues as a research front to this day. Accordingly, we will begin this section with a brief historical survey.

Not long after Newton formulated his Law of Gravitation it was realised that there was a profound problem with the application of the law to real situations. The discovery was soon made that it is possible to write down equations which, given the present state of two gravitating bodies, will describe their past and future motion with perfect accuracy. However, it was found that the same is not true for the case of three or more bodies. This became known as the N-body problem.

Astronomers reacted to the impasse in two ways. First, they concentrated on global properties, such as the total energy of a system or the trajectory of its centre of mass, for which they *were* able to derive precise formulas. Second, they considered systems like our Solar System, which can, under certain assumptions, be treated as modified two-body problems. Both of these lines of research were successful and *celestial mechanics* became an active field.

From the beginning, astronomers realised that they could, in principle, attack the N-body problem with numerical methods. That is, they were aware that the equations of motion for a set of bodies could be numerically integrated, over a series of small intervals, from the initial conditions to any desired point in the future. This technique was given the name *special perturbations* and it was applied to a few problems. However, most workers in celestial mechanics ignored it because of the long and laborious calculations required.

All this changed with the appearance of computers, which were able to perform arithmetic operations at high speed. Special perturbations, which works for any configuration of bodies, became the basic tool of workers in the field of dynamical astronomy. The technique has been refined and extended in many ways, only some of which have been used in Gravitas.

The purpose of this section is to describe some of the methods used in Gravitas, and some others that could be added to increase its speed and accuracy.

A.2 Fundamental Limitations

First, we should strike a note of caution. Miller (1964) discovered that all numerical simulations will ultimately fail. He first noted that the trajectories arising from two nearly identical sets of initial conditions diverge. This in itself was not unexpected as mathematicians realised from the outset that numerical errors accumulate as the integration process is extended. However, Miller found that the *rate* of divergence renders simulations of many systems unreliable: the properties that the dynamical astronomer is investigating are at risk of being submerged by the growing error. Miller's findings did not bury dynamical astronomy as a field, but they did place a health warning, as it were, on all numerical research. As Aarseth and Lecar put it:

> "Although this result has cast a shadow over N-body calculations ever since, it has not deterred subsequent investigations from being carried out in an optimistic spirit." (Aarseth and Lecar, 1975)

The principal generators of error, according to Miller, are close binary collisions, that is to say, encounters between two bodies which involve large accelerations and highly curved orbits. As Aarseth (1985) has shown, much effort has gone into dealing with these kinds of situation. However, we will begin by looking at the simplest techniques.

A.3 Aarseth's Basic Method

Although Aarseth was not the first person to state the formulae for the method of special perturbations, the basic equations have become associated with his name. In (Aarseth, 1962) he expressed the fundamental relation as:

$$\frac{d^2\mathbf{r}_i}{dt^2} = G \sum_{\substack{j=1 \\ j\neq i}}^{N} \frac{m_j (\mathbf{r}_i - \mathbf{r}_j)}{|\mathbf{r}_i - \mathbf{r}_j|^3}$$

In words, this differential equation (which we have simplified slightly) says that the acceleration of a mass i is equal to the sum of all the *other* masses, *each* divided by its separation from mass i, *all* multiplied by the Universal Constant of Gravitation.

This equation of motion is then integrated twice with respect to time to obtain the next position of the mass. The integration is carried out for each of the masses present so that *all* the positions are updated. Finally, the entire process is repeated for as long as desired.

A.4 Choice of Integration Method

The simplest form of numerical integration is known as Euler's method. In this scheme the differential equation is transposed into a *difference* equation and multiplied out to gain the next position of a mass. There are several ways in which this can be improved upon, and we will briefly review two that are commonly used. An excellent survey of numerical integration techniques is given in (Roy, 1978 Chapter 7).

The first constructs a more complex difference equation, adding terms which incorporate higher derivatives of the force terms. These terms compensate for the errors that are generated by the transposition of the differential equation. Because the terms are computed from a Taylor series expansion of the original equation of motion, this technique is known as the *polynomial* method.

The second method also constructs a more complex difference equation to increase precision. It adds four extra terms to compensate for the error in a step of integration. These terms derive from geometrical considerations of the way error accumulates and they lead to a *linear* difference equation. This technique is named the Runge-Kutta method, after its originators.

At present, Gravitas uses Euler's method. We have tried Runge-Kutta techniques but the increased precision did not provide enough benefit to outweigh disadvantages of speed and complexity. We have not tried the polynomial method as yet.

A.5 Previous Force Evaluations

Another method of improving the accuracy of the numerical process takes a strategic approach. Once the step by step integration process is going it is possible to use previous calculations of the force on a mass to compute a list of successive differences. These can then be used (as analogues for higher derivatives of the force function) in a difference equation structurally similar to the Taylor series polynomial mentioned above. Of course, this method cannot begin until several steps of the iteration have been carried out by some

other means. Nevertheless, it has become ubiquitous in the literature on dynamical simulations.

On the question of how many previous force evaluations it is worth preserving, Aarseth has written:

> "Practical considerations such as initialization, restarts, machine accuracy and increasing storage suggest that remembering four previous force evaluations is a good compromise." (Aarseth, 1985 p252)

Previous force evaluation has been used in Gravitas but was removed for reasons of expediency. It would require a great deal of programming effort to combine this method with some of the other techniques used to generate Massob behaviour. However, at some point in the future this would be a worthwhile task.

A.6 Individual Time Step

One way of reducing the computational overhead is called the *individual time step* method (Aarseth, 1971). This technique is based on the observation that it is not necessary to calculate to the same precision for every mass in a system. While some of the masses are undergoing violent accelerations, which require integration steps over short time intervals, others may be moving almost rectilinearly, or in stable orbits for which it is wasteful to use such small steps.

The individual time step method involves calculating a "reasonable" time step for each object in a system, using a criterion such as the relative change in force during the last step. The integration is then carried out over the shortest time step, but only for those objects with that step. Time is then advanced and the integration is carried out again, including, this time, any more objects whose time step has now been spanned. After several of these *auxiliary* steps, all the masses will have been advanced and the process can begin again.

The individual time step method has been tried in Gravitas. In some cases, for instance a simulation of the entire Solar System, it provided worthwhile improvements, while in others, such as a simulation of just the Terran planets (Mercury, Venus, Earth and Mars), the improvement was negligible. Again, for reasons of expediency, the individual time step method

is not used in Gravitas at present, but it would be considered for future versions.

A.8 Heuristic Methods

Besides improving the numerical methods there is another way in which the performance of Gravitas could be raised. For example, if only two Massobs are present, the program could use the analytic solution to the two-body problem and thereby work with perfect accuracy. Or it could examine the configuration of the Massobs in a more complex system and identify situations where the two-body solution is good enough.

There are a range of established techniques which do just this, but we do not have room to summarise them here. (Roy, 1978) contains a survey and (Aarseth, 1985) has discussions of restricted three-body techniques being applied in the same context. No heuristic methods have been used in Gravitas.

References

Aarseth, S.J. (1962), Dynamical Evolution of Clusters of Galaxies. *Monthly Notices of the Royal Astronomical Society.* **126** (3) 221-255

Aarseth, S.J. (1971), Direct Integration Methods of the *N*-Body Problem. *Astrophysics and Space Science.* **14** 118-132

Aarseth, S.J. (1985), Direct N-Body Calculations. In Goodman, J., and Hut, P. (Eds.), *Dynamics of Star Clusters.* IAU.

Aarseth, S.J., and Lecar, M. (1975), Computer Simulations of Stellar Systems. *Annual Review of Astronomy and Astrophysics.* **13** 1-21

Abelson, H., and diSessa, A.A. (1980), Turtle Geometry: The computer as a medium for exploring mathematics. MIT Press. Cambridge, MA.

Abelson, H., diSessa, A.A., and Rudolph, L. (1975), Velocity Space and the Geometry of Planetary Orbits. *American Journal of Physics.* **43**(7) 579-589.

Abelson, H., and Sussman, G.J. (1985), *The Structure and Interpretation of Computer Programs.* MIT Press. Cambridge, MA.

Adams, S.T. (1989), *Developing Databases and Knowledge Spaces with Boxer: An Illustration Based on Dinosaur Knowledge,* Technical Report G4. Available from The Boxer Group, Graduate School of Education, University of California, Berkeley, CA.

Adams, T. (1988), Computers in Learning: A Coat of Many Colours. *Computers and Education.* **12** (1) 1-6.

Baker, C.L. (1981), JOSS, Johnniac Open Shop System. In Wexelblat, R.L. (Ed.) *History of Programming Languages.* Academic Press. New York.

Baker, R.M.L. (1967), *Astrodynamics: Applications and Advanced Topics.* Academic Press. New York.

Bliss, J., Ogborn, J., Boohan, R., Briggs, J., Brosnan, T., Brough, D., Mellar, H., Miller, R., Nash, C., Rodgers, C., and Sakonidis, B. (1992), Reasoning Supported by Computational Tools. *Computers and Education* **18** (1-3) 1-9.

Bork, A.M. (1979), Interactive Learning. *American Journal of Physics.* **47** 5-10.

Brna, P. (1987), Confronting Dynamics Misconceptions. *Instructional Science* **16**, 351-379

Brna, P. (1989), Programmed Rockets: An Analysis of Students' Strategies. *British Journal of Educational Technology* **20** 1, 27-40

Brna, P. (1991), Promoting Creative Confrontations. *Journal of Computer Assisted Learning.* **7**, 114-122.

Brosnan, T. (1990), Using Spreadsheets in the Teaching of Chemistry. More Ideas and Some Limitations. *School Science Review*, **71** 256, 53-59.

Brown, J. S., Burton, R. R., and Bell, A. G. (1975), SOPHIE: A step towards a reactive learning environment. *International Journal of Man Machine Studies.* **7**.

Brown, J. S., Burton, R. R., and De Kleer, J. (1982), Pedagogical, Natural Language and Knowledge Engineering Techniques in SOPHIE I, II and III. In Sleeman, D. and Brown, J.S. (Eds.) (1982), *Intelligent Tutoring Systems.* Academic Press. London.

Brown, J. S. (1985), Idea Amplifiers - New Kinds of Electronic Learning Environments. In *Educational Horizons.* Spring 1985.

Bruner, J.S. (1968), *Toward a Theory of Instruction.* Harvard University Press. Cambridge, MA.

Burns, B., and Hagerman, A. (1989), Computer Experience, Self-Concept and Problem-Solving: the Effects of Logo on Children's Ideas of Themselves as Learners. *Journal of Educational Computing Research*, **5** (2) 199-212.

Cathcart, W.G. (1990), Effects of Logo on Cognitive Style. *Journal of Educational Computing Research.* **6**(2) 231-242.

Chang, S.K. (1987), Visual Languages: A Tutorial and Survey. *IEEE Software,* January 1987 29-39.

Clements, D.H., and Gullo, D.F. (1984), Effects of Computer Programming on Young Children's Cognition. *Journal of Educational Psychology.* **76**(6) 1051-1058.

Cox, B. (1986), *Object-oriented Programming: An Evolutionary Approach.* Addison-Wesley. New York.

Department of Education and Science (1991), *Science in the National Curriculum.* HMSO. London.

diSessa, A. A. (1982), Unlearning Aristotelian Physics: A Study of Knowledge-Based Learning. *Cognitive Science,* **6**(1) 37-75.

diSessa, A.A. (1986a), From Logo to Boxer, a new Computational Environment. *Australian Educational Computing*, July 1986.

diSessa, A. A. (1986b), Notes on the Future of Programming. In Norman, D.A. and Draper, S.W. (Eds.) *User Centred System Design*, Lawrence Erlbaum. Hillsdale, NJ.

diSessa, A. A. (1990), *Local Sciences: Viewing the Design of Human-Computer Systems as Cognitive Science*, Technical Report G6. Available from The Boxer Group, Graduate School of Education, University of California, Berkeley, CA.

diSessa, A. A., and White, B.Y. (1982), Learning Physics from a Dynaturtle. *Byte magazine* August 1982.

diSessa, A. A., and Abelson, H. (1986), Boxer: a Reconstructible Computational Medium. *Communications of the ACM*. **29**(9) 859-868.

Drescher, G. (1987), Object-Oriented Logo. In Lawler, R.W. and Yazdani, M. (Eds.) *Artificial Intelligence and Education, Volume One*. Ablex. Norwood, NJ.

du Boulay, J. B.H. (1978), *Learning Primary Mathematics Through Computer Programming*. Ph.D Dissertation. Department of Artificial Intelligence. Edinburgh University.

Eichenbaum, L., and Orbach, S. (1982), *Outside In... Inside Out*. Penguin Books. London.

Elsom-Cook, M. (Ed.) (1990), *Guided Discovery Tutoring*. Paul Chapman. London.

Eyler, M.A. (1990), Simulation Modeling Using Spreadsheets. Academic Computing on Macintosh Environment, Bogaziçi University Publication No. 472.

Feurzeig, W., Papert, S., Bloom, M., Grant, R. and Solomon, C (1969), *Programming Languages as a Conceptual Framework for Teaching Mathematics*. BBN Report No. 1889. Bolt, Beranek and Newman Laboratories, Cambridge, MA.

Feurzeig, W. (1984), The Logo Lineage. In Ditla, S. (Ed.) *Digital Deli*. Workman Press. New York.

Foley, J.D., and Van Dam, A. (1982), *Fundamentals of Interactive Computer Graphics*. Addison-Wesley. New York.

Forman, G., and Pufall, P.B.(Eds.) (1988), *Constructivism in the Computer Age.* Lawrence Erlbaum. Hillsdale, NJ.

Frasson, C. and Gauthier, G. (Eds.) (1990), *Intelligent Tutoring Systems.* Ablex, Norwood NJ.

Galizia, M.T. (1990), Experiences of Computer Laboratory in Mathematics Teaching. In *Proceedings of the NATO Conference on Advanced Technologies in the Teaching of Mathematics and Science.* Milton Keynes.

Gettys, W.E., Keller, F.J., and Skove, M.J. (1989), *Physics, Classical and Modern.* McGraw-Hill, New York.

Goldberg, A. and Robson, D. (1983), *Smalltalk-80: the language and its implementation.* Addison-Wesley. New York.

Goldenberg, E.P. (1982), Logo: A Cultural Glossary. *Byte magazine* August 1982.

Goldstein, I.P (1975), Summary of MYCROFT: A System for Understanding Simple Picture Programs. *Artificial Intelligence.* **6** (3) 249-288

Graham, I., (1991), *Object Oriented Methods.* Addison Wesley. London.

Groen, G. (1984), Theories of Logo. In Sorkin, R. (Ed.) *Logo 84: Pre-proceedings.* MIT.

Hammond, N., and Trapp, A. (1992), Matching CBL Approach to Learning Need: A Heuristic Methodology for Instructional Design. In Brusilovsky, P., and Stefanuk, V. (Eds.) *Proceedings of the East-West Conference on Emerging Computer Technologies in Education.* Moscow.

Harel, I., and Papert, S. (1990), Software Design as a Learning Environment. *Interactive Learning Environments.* **1** 1-32.

Hodgkin, R. A. (1980), Mountains and Education. *The Alpine Journal.* **86** (330) 201-11.

Hodgkin, R. A. (1985), *Playing and Exploring.* Methuen. London.

Hollan, J.D., Hutchins, E.L., and Weizman, L. (1984) STEAMER: an interactive inspectable simulation-based training system. *AI Magazine,* **5** (2).

Holm, P.E. (1988), Petrogenetic Modeling with a Spreadsheet Program. *Journal of Geological Education.* **36** (3).

Howe, J., O'Shea, T.M.M., and Plane, F. (1979), Teaching Mathematics Through Logo Programming: An Evaluation Study. In In Lewis, R. and Tagg, D.

(Eds.) *Computer-Assisted Learning – Scope, Progress and Limits.* North-Holland. Amsterdam.

Howe, J., Ross, P., Johnson, K., Plane, F., and Inglis, R. (1982), Teaching Mathematics Through Programming in the Classroom. *Computers and Education* **6** p85-91

Hughes, M. and MacLeod, H. (1986), The Craigmillar Logo Project. In Lawler, R., du Boulay, B., Hughes, M. and Macleod, H., *Cognition and Computers: Studies in Learning.* Ellis Horwood. Chichester.

Hutchins, E.L., Hollan, J.D., and Norman, D.A. (1986), Direct Manipulation Interfaces. In Norman, Donald A. and Draper, Stephen W. (Eds.), *User Centred System Design.* Lawrence Erlbaum. Hillsdale, NJ.

Jeans, J. (1947), *The Growth of Physical Science.* Cambridge University Press.

Jeans, J. (1967), *An Introduction to the Kinetic Theory of Gases.* Cambridge University Press.

Klotz, L.L. (1989), *Boxer: The Programming Language,* Unpublished B.S. Thesis, Massachussetts Institute of Technology, Cambridge, MA.

Knowledge Revolution. (1989), *Interactive Physics.* Software for the Apple Macintosh. Knowledge Revolution, San Mateo, CA.

Lawler, R.W. (1979), *One child's learning.* PhD Dissertation. Department of Electrical Engineering and Computer Science. MIT, Cambridge MA.

Lawler, R.W. (1982), Designing Computer-Based Microworlds. *BYTE magazine.* August 1982.

Lawler, R.W. (1985), *Computer Experience and Cognitive Development.* Ellis Horwood. Chichester.

Lawler, R.W. (1987), Learning Environments: Now, Then, and Someday. In Lawler, R. L. and Yazdani, M. (Eds.) *Artificial Intelligence and Education, Volume One.* Ablex. Norwood, NJ.

Lecar, M. and Aarseth, S.J. (1986), A Numerical Simulation of the Formation of the Terrestrial Planets. *The Astrophysical Journal.* **305** 564-579.

Leron, U. (1985), State Transparency and Conjugacy. *Micromath.* Winter 1985.

Martin, F. and Resnick, M. (1990), LEGO / Logo and Electronic Bricks: Creating a Scienceland for Children. In *Proceedings of the NATO Conference on*

Advanced Technologies in the Teaching of Mathematics and Science. Milton Keynes.

Mellar, H. (1989), Creating Alternative Realities: Computers, Modelling and Curriculum Change. In Hoyles, C. and Noss, R. (Eds.), *Mathematics versus the National Curriculum.* Farmer Press.

Miller, R.H. (1964), Irreversibility in Small Stellar Dynamical Systems. *The Astrophysical Journal* **140** (1) 250-256

Miller, R.S.; Ogborn, J.M.; Turner, J.; Briggs, J.H. and Brough, D.R. (1990), Towards a Tool to Support Semi-Quantitative Modelling. In *Proceedings of the International Conference on Advanced Research on Computers and Education.* Tokyo.

Millwood, R. and Stevens, M. (1989), What is the Modelling Curriculum? In Kibby, M. (Ed.), *Computer Assisted Learning: Selected Proceedings from the CAL '89 Symposium.* Pergamon Press. Oxford.

Minsky, M. and Papert, S. (1972), The '72 Progress Report. MIT AI Lab Memo.

Minsky, M. (1975), A framework for representing knowledge. in P. H. Winston, *The Psychology of Computer Vision.* McGraw-Hill. New York.

Myers, B.A. (1986), Visual Programing, Programming by Example, and Program Visualization: A Taxonomy. *Conference Proceedings, CHI '86: Human Factors in Computing Systems.* Association for Computing Machinery.

Nelson, T. H. (1967), Getting it out of our system. In G. Schechter (Ed.), *Information Retrieval: A Critical Review* (pp 191-210). Thompson. Washington, DC.

Newcombe, A. and Stewart, K. (1985), Exploring in a Physics Microworld The Laws of Gravity and Motion. In Palmgren, M. (Ed.), *Logo 85: Pre-proceedings.* MIT, Cambridge MA.

Ogborn, J. and Wong, D. (1984), A Microcomputer Dynamical Modelling System. *Physics Education,* **19** 138-142.

Oke, K.H. and Jones, A.L. (1982), Mathematical Modelling in Physics and Engineering- part 1. *Physics Education,* **17** 220-3

O'Shea, T. and Smith, R. (1987), Understanding Physics by Violating the Laws of Nature: Experiments with the Alternate Reality Kit. *Proceedings of the Conference on Computer-Assisted Learning (CAL '87)*

Palmgren, M. (1985), *Logo 85: Pre-proceedings*. MIT, Cambridge MA.

Papert, S. (1970), Teaching Children Thinking. Paper delivered to the 1970 IFIP Conference on Computer Education. In Soloway, E., and Spohrer, J.C. (Eds.) *Studying Novice Programmers*. Lawrence Erlbaum. Hillsdale NJ.

Papert, S., and Solomon, C. (1971), Twenty Things to do with a Computer. *Logo Memo No.3*. MIT. Reprinted in Soloway, E., and Spohrer, J.C. *Studying Novice Programmers*. Lawrence Erlbaum. Hillsdale, NJ.

Papert, S. (1972), Teaching Children to be Mathematicians Versus Teaching About Mathematics. *Int. J. Math. Educ. Sci. Technol.* **3** 249-262.

Papert, S.; Watt, D.; diSessa, A.A. and Weir, S. (1979), *Final Report of the Brookline Logo Project*. Logo Memos 53 & 54. MIT, Cambridge MA.

Papert, S. (1980), *Mindstorms, Children, Computers and Powerful Ideas*. Harvester Press. Brighton.

Papert, S. (1987a), Microworlds: Transforming Education. In Lawler, R. L. and Yazdani, M. (Eds.) *Artificial Intelligence and Education, Volume One*. Ablex. Norwood, NJ.

Papert, S. (1987b), Computer Criticism versus Technocentric Thinking. *Educational Researcher*, **16** 22-30. Also published in *MIT Logo 85 Conference: Theoretical Papers*.

Paradigm Software Inc. (1990), *Object Logo*. Cambridge, MA.

Pea, R.D., Hawkins, J., and Sheingold, K. (1983), *Developmental Studies on Learning Logo Computer Programming*. Paper presented to the Biennial Meeting of the Society for Research in Child Development. Detroit, MI.

Ploger, D. and Carlock, M. (1991), Programming and Problem Solving: Implications for Biology Education. *Journal of Artificial Intelligence in Education*. **2** (4).

Polanyi, M. (1962), *Personal Knowledge*. Routledge and Kegan Paul. London.

Resnick, M. and Ocko, S. (1990), LEGO / Logo: Learning Through and About Design. In Harel, I. (Ed.) *Constructionist Learning*. MIT Media Laboratory.

Richmond, B., Peterson, S., and Vescuso, P. (1987), *STELLA*. High Performance Systems Inc.

Ross, P., and Howe, J. (1981), Teaching Mathematics Through Programming: Ten Years On. In Lewis, R. and Tagg, D. (Eds.) *Computers in Education:*

Preprints to the World Conference on Computers and Education, Lausanne 1981. North-Holland. Amsterdam.

Roy, A.E. (1978), *Orbital Motion.* Adam Hilger, Bristol.

Schecker, H. (1990), The Didactic Potential of Computer Aided Modeling for Physics Education. In Ferguson, D.L. (Ed.) *Advanced Technologies in the Teaching of Mathematics and Science.* Springer-Verlag. London.

Shneiderman, B. (1982), The Future of Interactive Systems and the Emergence of Direct Manipulation. *Behaviour and Information Technology.* **1** 237-256

Shneiderman, B. (1983), Direct Manipulation: a Step Beyond Programming Languages. *IEEE Computer,* **16** (8) 57-63

Shu, N.C. (1986), Visual Programming Languages: A Perspective and a Dimensional Analysis. In Chang, S.K., Ichikawa, T., and Ligomenides, P.A. (Eds.) *Visual Languages.* Lawrence Erlbaum. Hillsdale, NJ.

Shute, V., and Bonar, J. (1986), Intelligent Tutoring Systems for Scientific Enquiry Skills. In *The Proceedings of the Eighth Annual Conference of the Cognitive Science Society.* Lawrence Erlbaum. Hillsdale, NJ.

Shute, V., and Glaser, R. (1990), A Large-Scale Evaluation of an Intelligent Discovery World: Smithtown. *Interactive Learning Environments,* **1** 51-77.

Shulman, S.L. and Keislar, E.R. (1966), *Learning by Discovery. A Critical Appraisal.* Rand McNally. Chicago IL.

Sinclair, G. and Colton, M. (1985), Ideamap: An Idea Composing Microworld. In Palmgren, M. (1985), *Logo 85: Pre-proceedings.* MIT, Cambridge MA.

Sleeman, D. and Brown, J.S. (Eds.) (1982), *Intelligent Tutoring Systems.* Academic Press. London.

Smith, R. B. (1986), The Alternate Reality Kit: An Animated Environment for Creating Interactive Simulations. In *Proceedings of the IEEE Computer Society Workshop on Visual Languages.*

Smith, R. B. (1987), Experiences with the Alternate Reality Kit: An Example of the Tension between Literalism and Magic. In *Proceedings of the Computer Human Interaction and Graphical Interface Conference.* Toronto.

Solomon, C., and Papert, S. (1976), A Case Study of a Young Child Doing Turtle Graphics. *Logo Memo No.28.* MIT 1976.

Solomon, J.C. (1962), Fixed Idea as an Internalised Transitional Object. *American Journal of Psychotherapy.* **16.**

Spensley, F., O'Shea, T., Singer, R., Hennessey, S., O'Malley, C. and Scanlon, E. (1990), An 'Alternate Realities' Microworld for Horizontal Motion. *CITE Report No. 105,* Centre for Information Technology in Education, Open University, Milton Keynes.

Squires, D. and McDougall, A. (1986), Computer-based Microworlds - A Definition to Aid Design. *Computer Education,* **10** (3) 375-378.

Squires, D. and Sellman, R. (1985), Designing Computer Based Microworlds, In Palmgren, M. (1985), *Logo 85: Pre-proceedings.* MIT, Cambridge MA.

Statz, J. (1973), *The Development of Computer Programming Concepts and Problem-Solving Abilities among Ten-Year-Olds Learning Logo.* Unpublished PhD Thesis, Syracuse University (0659).

Swan, K (1991), Programming Objects to Think With: Logo and the Teaching and Learning of Problem Solving. *Journal of Educational Computing Research,* **7** (1) 89-112.

Tatar, D. (1987), *A Programmer's Guide to COMMON LISP.* Digital Press. Bedford, MA.

Taylor, R. (Ed.) (1980), *The Computer in the School: Tutor, Tool, Tutee.* Teachers College Press.

Teodoro, V.D. (1990), The Computer as a Conceptual Lab: Learning Dynamics with an Exploratory Environment. In Ferguson, D.L. (Ed.) *Advanced Technologies in the Teaching of Mathematics and Science.* Springer-Verlag. London.

Thompson, P. W. (1985a), A Piagetian approach to Transformation Geometry via Microworlds. *The Mathematics Teacher.* September 1985.

Thompson, P. W. (1985b), Experience, problem-solving and learning mathematics: Considerations in developing Mathematics curricula. In E.A. Silver (Ed.), *Learning and teaching mathematical problem solving: Multiple research perspectives.* Lawrence Erlbaum. Hillsdale, NJ.

Thompson, P. W. (1987), Mathematical Microworlds and Intelligent Computer Assisted Instruction. In: Greg P. Kearsley (Ed.), *Artificial Intelligence and Instruction: Applications and Methods*. Addison-Wesley.

Turkle, S. and Papert, S. (1990), Epistemological Pluralism: Styles and Voices Within the Computer Culture. *Signs*, **16** (1) 129-157

Wenger, E. (1987), *Artificial Intelligence and Tutoring Systems*. Morgan Kaufmann. Los Altos CA.

White, B. (1984), Designing Computer Games to Help Physics Students Understand Newton's Laws of Motion. *Cognition and Instruction*. **1** (1) 69-108.

Winnicott, D.W. (1951), *Transitional Objects and Transitional Phenomena*. Tavistock Publications. London.

Winnicott, D.W. (1971), *Playing and Reality*. Tavistock Publications. London.

Winograd, T. (1972), *Understanding Natural Language*. Academic Press.